DESIGN AND IMPLEMENTATION OF A LOW POWER ASYNCHRONOUS GPS BASEBAND PROCESSOR

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Stephen James Longfield Jr

May 2013

ABSTRACT

This thesis presents the design and implementation of a low-power asynchronous circuit for processing the digital component of the Global Positioning System (GPS) signal. The GPS signal structure is described and opportunities for applications of asynchronous design are presented. A selection of interesting circuit implementations exploiting these opportunities are presented and their efficiency and correctness are argued. The thesis continues with a description of how these circuits were implemented in silicon circuits and concludes with a comparison of power and performance between this implementation and other contemporary GPS systems. This comparison finds the 1.4 mW GPS processor described to use less energy per position update than any published processor.

ACKNOWLEDGEMENTS

I would like to thank Rajit Manohar for his assistance in asynchronous design, and for his willingness to give me the freedom to find my own path. I would like to thank Benjamin Tang and Abin Thomas for their work on this project, and the late Paul Kintner for his advice on GPS systems. I cannot thank Robert Karmazin and Carlos Tadeo Ortega Otero enough for their work on CellTK and LayoutTK—without these tools, the physical implementation of the circuits presented here would have been excessively painful—I hope they can forgive me for all the pestering over these past few years. Finally, I would like to thank my parents and sister for their support and encouragement.

This material is based upon work supported in part by the National Science Foundation Graduate Research Fellowship under grant No. DGE-0707428.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In this section, we motivate the need for low-power baseband processing and give an introduction of contemporary work. It concludes with an overview of the organization of this thesis.

1.1 Motivation

The use of Global Positioning System (GPS) technology is ever increasingly affecting our lives. We rely on GPS to navigate from place to place, to locate people and objects, to provide time synchronization in our telecommunication networks and power grids, and in many other everyday applications.

Today, high power consumption of existing GPS receiver chips can cause overheating issues, and can limit continuous GPS operation in mobile devices. It is clear that the high power consumption in GPS receivers must be addressed to pave the way for advances in areas such as location-aware applications and micro-robotics navigation.

Asynchronous techniques enable very low-power designs, especially in systems where the rate of required throughput may vary over time [3], [25], [14]. As a GPS system involves several different components, each of which compute at a different natural frequency, an asynchronous design could lead to benefits in power consumption for the baseband processing of GPS signals.

A typical GPS receiver consists of an Radio Frequency (RF) front end and a Digital Signal Processor (DSP). The RF front end receives the GPS signal from the satellites, mixes it down to an intermediate frequency, and samples it. The DSP acquires a lock to multiple GPS satellite signals present in the front end samples and tracks variations in the signals over time. While the DSP tracks variations in the signal, it also extracts information from it that can be used to compute the current position and time—the "navigation solution."

The power consumption of this DSP, or of alternative baseband processing techniques, typically dominates the RF power, making it an obvious place to concentrate power optimization.

1.2 Related Work

As low power consumption is of obvious benefit, there has been substantial research into low power GPS systems and low power asynchronous systems, using a variety of methods. Presented here is a selection of interesting low power GPS systems, some of which use very different methods than are given in the text, and a selection of low power asynchronous systems.

1.2.1 Low Power GPS Systems

Much of the work on low power GPS systems focuses on innovative signal processing techniques, or on using additional hardware to avoid certain power-hungry operations.

For example, the work of Namgoong et. al. [19] the system is duty cycled, spending the majority of its time powered off. This system is functional due to the high speed of the satellite acquisition phase, which has been implemented in a highly parallel fashion (42 correlators per channel, of which there are 5 on chip). By spending the majority of its time powered off it does not provide continuous tracking information. Additionally, this system is higher latency than conventional methods, and so its usefulness is limited to low velocity systems where significant delay does not add significant position error. It does have very low-power operation, and is capable of reporting position with less than 7m of RMS error given a velocity of less than 30 m/s while consuming only 4 mW. Note also that this system does not collect the data that is in the GPS signal, only the relative timing offsets between signals, and so must be augmented with external information (as in the original publication)

or another receiver to complete the position tracking.

This approach is similar to work done by Meng et. al. [17] where a copy of the satellite data is shifted into a register before correlation. In general, these methods reduce power consumption, but have a large latency and area overhead and require a secondary data source. This approach has not found significant adoption in commercial GPS systems. Both of these systems have the RF front end separate from the GPS processor, to avoid the significant Electro-Magnetic Interference (EMI) that is caused by their synchronous design's switching nose.

The more traditional approach to reducing power in GPS receivers can be found in the work of Wei et. al. [30] and Gramegna et. al. [7], which both use System-on-Chip (SoC) based solutions with a synchronous processor doing the acquisition and tracking of the signals. In both of these publications, the SoC uses approximately two thirds of the power consumed, and the focus is on reducing the power used by the satellite radio receiver. As this radio power has been brought below 400μW by some modern designs [8], but the total power consumption is still above 10 mW for the best designs available[1] there exists quite a bit of room for improvement in the baseband data processing. These, and other similar SoC based solutions tend to be based on modified off-the-shelf hardware that is capable of tracking more satellites than are ever visible [30], and thereby may use more power than they would otherwise require.

An approach similar to the duty cycled systems was recently proposed by Liu et. al. [12]. In particular, it was proposed as having applications for mobile devices, particularly cell phones, as well as possible application in animal tracking devices, when continuous operation may be unimportant. This system collects a small amount of GPS data, enough to get a coarse lock on the signal, transmits that data over a mobile data connection, and computes the location in the cloud. As it does not have continuous tracking information, there is a

[1]Broadcom BCM2075,
http://www.gpsworld.com/wp-content/uploads/2012/10/2012GPSWorld_Receiver_Survey.pdf

limit to the attainable precision, resulting in large average errors of 43 meters, and very large maximum errors of 350 meters. Similar to the duty cycled solutions above, this system requires alternate methods to obtain the satellite data, though as the National Oceanic and Atmospheric Association makes this data freely available on the internet[2], this does not provide a significant barrier for cloud-offloaded computation. Again similarly to the duty cycled solutions, the power consumption depends on the duty cycle, and is given as consuming approximately 0.5 mJ per data point, though this does not include the energy to transmit or receive data from the cloud service, and is only being calculated very intermittently.

1.2.2 Low Power Asynchronous Systems

QDI asynchronous systems have been found to generate lower power circuits in many different applications, but particularly in applications where the typical activity pattern is occasionally interrupted by complex operations [28]. This makes intuitive sense, as while a synchronous system demands that all actions are bound by the worst case delay of any subsystem, an asynchronous system does not require that the worst case binds all other cases [14].

These properties of asynchronous systems have been leveraged to reduce power in floating point units (FPU) [25], where there are many uncommon, complex cases. These cases typically consume a disproportionate amount of power, area, and design time for synchronous FPUs, but asynchronous design enables simpler implementations. Similar optimizations, combined with other techniques enabled by an asynchronous implementation style have been found to reduce power and increase throughput in FPGAs [27] and microprocessors [5].

[2] www.ngs.noaa.gov/orbits/

1.3 Core Contribution

This thesis focuses on the contributions that asynchronous design had in reducing the power consumption of baseband GPS processing. It describes circuits that may be useful in reducing the power consumption of other signal processing applications. The thesis describes the techniques and tools that were used to implement these in QDI asynchronous circuits, focusing on the cases where these tools were adapted or developed to fit the particular demands of the application. As this thesis is based on the work of multiple contributers, the material here concentrates on the author's work. A discussion of some of the other contributions can be found in the first publication of this work [26].

1.4 Organization of Thesis

The GPS signal structure, and opportunities for asynchronous design are presented in Chapter 2. The design and implementation of the asynchronous baseband processor is discussed in Chapters 4 and 5. Performance metrics are given in Chapter 6, and Chapter 7 presents some concluding remarks.

CHAPTER 2

GLOBAL POSITIONING SYSTEMS

This section begins with a history of satellite navigation systems in the United States, and continues with a description of their general operation. It then focuses on the GPS system, giving both specifics of the implementation, and indications for where asynchronous design techniques may be advantageous.

2.1 Satellite Navigation Systems

There are several different satellite navigation systems that have been attempted over the years. The majority of them have been based on the same principles and have come from similar—military research—origins.

2.1.1 GPS History

The Global Positioning System project began in 1973 with the unofficial motto [21]:

> The mission of this Program is to:
>
> - Drop 5 bombs in the same hole, and
> - Build a cheap set that navigates (<$10,000), and don't you forget it!

The first part of the motto was satisfied by initial testing in the late 1970s and early 1980s, where low visibility bombing tests guided by GPS far outperformed those guided by radar. The usefulness of the system was further demonstrated during the First Gulf War, where the GPS was used for significant strategic advantage [21]. The cost of GPS receivers was initially quite high, but was first driven down when President Reagan issued a directive guaranteeing free use of GPS for civilian applications in 1983. This development initially focused on aircraft guidance [22] and was the starting point for international commercial development on low cost civilian receivers.

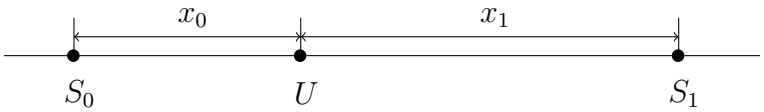

Figure 2.1: One Dimensional Distance Calculation

The technology has come quite a long way since then. We are now at the point where a commercial GPS system can easily find its position anywhere in the world with only a few meters of error. These systems are used in our cellular phones, for car navigation, and in a variety of robotics applications, and the costs have dropped significantly.

2.1.2 Basic Operation

The central idea behind GPS navigation is that the position of a certain point in space can be calculated by knowing the distance from this point to other, known positions. In one dimension, this can be considered as shown in Figure 2.1. If the positions of S_0 and S_1 are known, as well as the distances x_0 and x_1, it is trivial to calculate the position of U. Note that knowing the location of S_0 and the distance x_0 is not sufficient if we do not know if U is to the left or to the right of S_0; S_1 and x_1 are required to break the ambiguity. If there is non-systemic error in these measurements, $S_0 + x_0$ may disagree with $S_1 + x_1$. In this case, the average of the two candidate positions will likely have less error than they had individually.

The two dimensional case is shown in Figure 2.2. In this case, each position, S, and distance measurement, x, defines a circle on which U may reside. The intersection of these three circles is a point, which gives the position of U. This generalizes to the three dimensional case, where each of these circles is extended into spheres.

The real satellite case is similar to the image shown in Figure 2.2, where the position information from the satellites is given by ephemeris data, transmitted from the satellites themselves or collected from an online database. Distance from the satellites to the user is

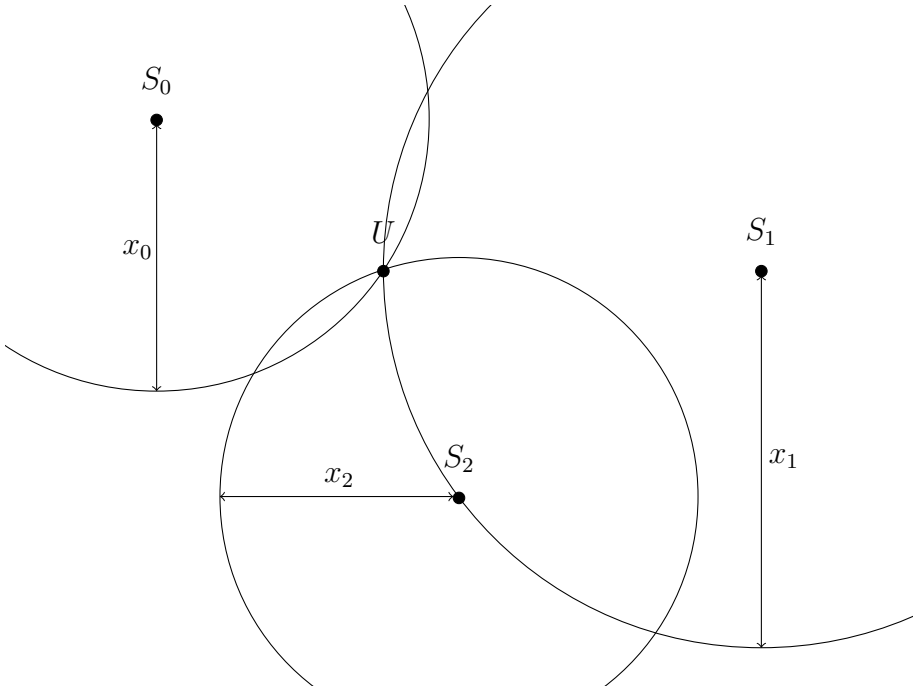

Figure 2.2: Two Dimensional Distance Calculation

calculated using differences between the send time (encoded in the satellite data), and the received time. To accurately determine the distance from these times, the users and satellites must have a very consistent knowledge of the current time, as the signals propagate at the speed of light; accumulating approximately 30 cm of error per nanosecond.

To avoid the need for costly atomic clocks in GPS receivers, most devices track 4 satellites, and solve for the time as well as position. Each one of the GPS satellites contains an atomic clock, synchronized daily to a reference clock in Boulder, Colorado. These clocks have an optional dither, known as "Selective Availability", which allows the U.S. Government to reduce the accuracy of GPS navigation, though it has not been enabled since May 2000, and the U.S. Government has stated that there are no plans to include this feature in future versions of GPS satellites [20].

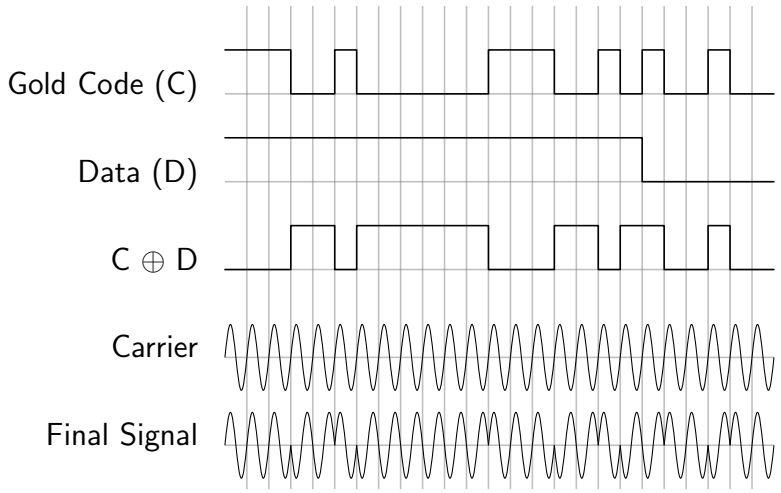

Figure 2.3: BPSK Modulation of GPS Signal

2.2 GPS Signal Structure

To put the later design decisions in context, this section contains a description of the GPS signal structure. There are actually several different signals available, however, this design and description focus on the L1 Coarse/Acquisition (C/A) signal, as details required to use the encrypted precision code (P(Y)) are classified, and the next generation satellites (GPS III) are not due to launch until 2014 [2] [21].

The L1 GPS signal is transmitted using a Code Division Multiple Access (CDMA) methodology, to allow multiple Space Vehicles (SVs) to communicate on a single carrier channel, specifically 1.57542 GhZ. This, and the other signals defined in the GPS protocol (L2, L5) are known as the L signals, as they occupy the 1 to 2 GhZ band known as the IEEE L-Band, and as a shorthand for "Link" [21].

The actual signal has three components, the CDMA code, the data and the carrier signal. They are encoded using a Binary Phase Shift Keying (BPSK) methodology, as shown in Figure 2.2. In this methodology, the data being sent from the SVs is XOR'd with the satellite's assigned code, and this is used to modulate the carrier.

The CDMA codes chosen are a particular class of codes, known as Gold codes, which

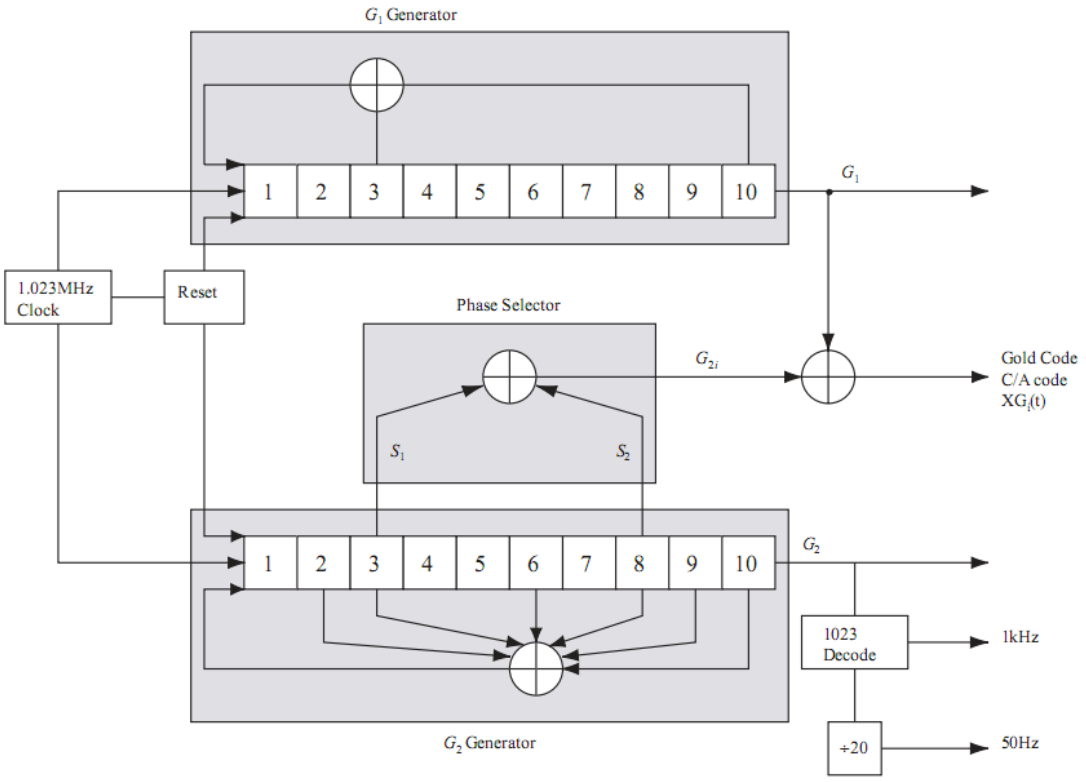

Figure 2.4: LFSR to generate Pseudorandom Gold code for SV31 [2]

are designed such that they can be generated by a Linear Feedback Shift Register (LFSR) Pseudo-Random Noise (PRN) generator, as shown in Figure 2.4, and have very good autocorrelation properties [6]. These codes are generated at a rate of 1.023 MHz, and are each 1023 'chips' long, meaning complete code generation happens at a rate of 1 KHz. As data received from the SVs is transmitted at a rate of 50 bits per second, twenty repetitions of the code are used to represent a single bit, to assist usage in very low signal strength environments.

Table 2.2 gives the assignment between codes and SVs. As each code is only assigned to a single SV, determining which codes are present in the received signal is identical to determining the SVs above the horizon.

Figure 2.5 gives an example of correlation between a simulated signal from SV1 and

Table 2.1: GPS Code Phase assignments [21]

SV ID No.	GPS PRN Number	Code Phase Selection G_2	Code Delay Chips	First 10 chips octal
1	1	$2 \oplus 6$	5	1440
2	2	$3 \oplus 7$	6	1620
3	3	$4 \oplus 8$	7	1710
4	4	$5 \oplus 7$	8	1744
5	5	$1 \oplus 9$	17	1133
6	6	$2 \oplus 10$	18	1455
7	7	$1 \oplus 8$	139	1131
8	8	$2 \oplus 9$	140	1454
9	9	$3 \oplus 10$	141	1626
10	10	$2 \oplus 3$	251	1504
11	11	$3 \oplus 4$	252	1642
12	12	$5 \oplus 6$	254	1750
13	13	$6 \oplus 7$	255	1764
14	14	$7 \oplus 8$	256	1772
15	15	$8 \oplus 9$	257	1775
16	16	$9 \oplus 10$	258	1776
17	17	$1 \oplus 4$	469	1156
18	18	$2 \oplus 5$	470	1467
19	19	$3 \oplus 6$	471	1633
20	20	$4 \oplus 7$	472	1715
21	21	$5 \oplus 8$	473	1746
22	22	$6 \oplus 9$	474	1763
23	23	$1 \oplus 3$	509	1063
24	24	$4 \oplus 6$	512	1706
25	25	$5 \oplus 7$	513	1743
26	26	$6 \oplus 8$	514	1761
27	27	$7 \oplus 9$	515	1770
28	28	$8 \oplus 10$	516	1774
29	29	$1 \oplus 6$	859	1127
30	30	$2 \oplus 7$	860	1453
31	31	$3 \oplus 8$	861	1625
32	32	$4 \oplus 9$	862	1712
—[1]	33	$5 \oplus 10$	863	1745
—[1]	34[2]	$4 \oplus 10$	950	1713
—[1]	35	$1 \oplus 7$	947	1134
—[1]	36	$2 \oplus 8$	948	1456
—[1]	37[2]	$4 \oplus 10$	950	1713

[1] Codes 33 through 37 are reserved for non-satellite use (e.g. ground transmitters)
[2] Codes 34 and 37 are the same

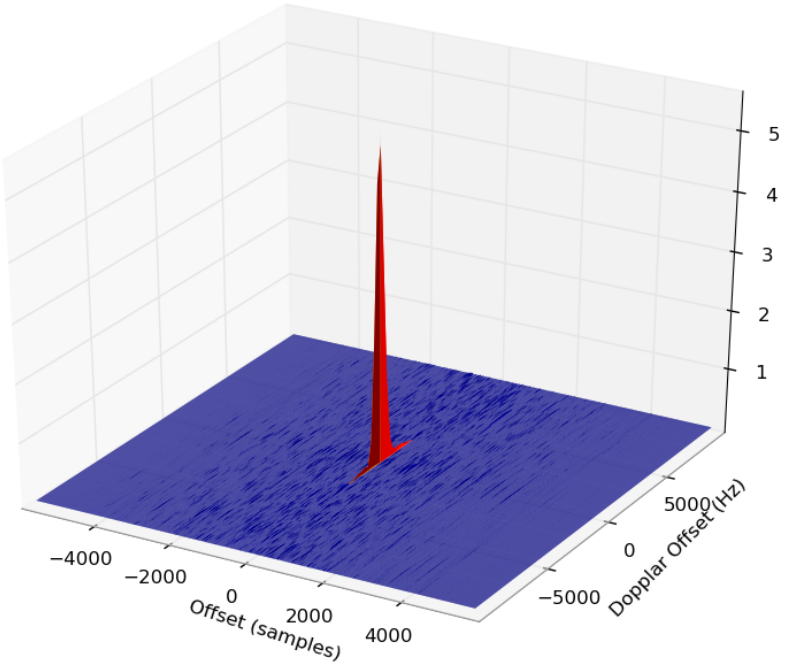

Figure 2.5: Correlation Of SV 1 C/A Code

several possible replicated signals. The replicated signals range over a variety of Doppler shifts (caused by the relative motion of the sending SV and receiver) and offsets on the generated Gold code. As can be seen from this diagram, the alignment peak is very narrow in the offset dimension, but wider in the Doppler offset dimension. Additionally, note that there are no 'false peaks', an important property of Gold codes.

The signal in Figure 2.5 was generated assuming a sampling rate of 5.714 MhZ, and with the received satellite signal signal mixed down to an intermediate frequency (IF) of 1.405 MhZ. This sampling rate and IF are are typical, and have a convenient property in that they are coprime. If they were multiples of each other (for instance, if the sampling frequency was exactly four times the IF), any alignment error that was present at startup will always be present. However, since they are different and do not have any common multiples, the sub-sample alignment will change between samples, and the rate at which it changes will allow for finer sub-sample resolution. Sub-sample resolution is important, as position is

Subframe 1:	TLM	HOW	Clock Corrections and SV heath/accuracy
Subframe 2:	TLM	HOW	Ephermis parameters
Subframe 3:	TLM	HOW	Ephermis parameters
Subframe 4*:	TLM	HOW	Almanac, ionopsheric model, dUTC
Subframe 5*:	TLM	HOW	Almanac

* Subframes 4 and 5 have 25 version before repeating

Figure 2.6: GPS Navigation Data Structure

calculated from differences in correlated SV timing, and at 5.714 MHz, a single sample error (approximately 175 nanoseconds) corresponds to a roughly 50 meter position error.

The GPS data packet is shown in Figure 2.6. Each subframe is 300 bits long, and there are 5 subframes in each frame [2]. At a datarate of 50 bits per second, this means a subframe is transmitted every six seconds, a full frame in 30 seconds, and, with subframes 4 and 5 spread out over 25 frames, the full GPS data packet being transmitted in 12.5 minutes. Each subframe contains the Telemetry (TLM) and Handover Word (HOW) pair. The TLM word is 24 bits long, beginning with the preamble sequence (10001011) and containing parity information. Next is the 17-bit HOW data, which contains the Time of Week (TOW), given in six-second (one subframe) increments.

The remainder of the data in the frames is not used directly in the baseband processing, but instead gives information about the ephermis (orbit) of the SVs, and error corrections for clock and atmospheric problems. This SV error correction information is updated once every eight hours by ground-truth base stations [21].

2.3 Opportunities for Asynchronous Design

The GPS signal structure and processing needs suggest some areas for low power asynchronous design. In particular, the fact that the sampling frequency and the data frequency are coprime implies coprime processing domains. These domain crossings can be difficult to implement and prone to metastability in synchronous designs, but can be implemented without difficulty in a QDI asynchronous design. Considering them as clock domains puts some timing and performance restrictions on the asynchronous design, however as they are relatively low-frequency, these performance requirements do not realistically constrict the design.

There are several pieces of infrequently performed, complex mathematical functions that need to occur over the course of tracking the signal. This type of complex, infrequent operation is the perfect application for QDI asynchronous VLSI, as by enabling average-case operation, infrequent high-complexity operations do not significantly affect the performance [25]. The more common operations, such as the accumulation of the data bit over the full Gold code or the numerically controlled oscillators used to reconstruct the IF carrier signal, can also be improved through asynchronous VLSI using specialized low-level optimizations.

Beyond the common operations, there are several infrequent operations, such as down-sampling the twenty repetitions of each data bit, which can be implemented using very simple hardware that would not necessarily meet the frequency requirements of the most demanding subsystems, but easily meet the very low frequency requirements of others.

CHAPTER 3

SYSTEM OVERVIEW

In this section, we will give an overview of how the GPS receiver system described in this thesis acquires a lock onto GPS SV signals and then tracks them as they vary over time.

The system expects 1-bit samples from an radio front end which it uses to acquire, track, and extract crucial data. After the data is extracted, it is sent to a co-processor or base station for navigation solution computation. In order to solve the four unknowns corresponding to the receiver time, longitude, latitude, and altitude position, a GPS receiver needs to track at least four satellites. To allow for redundancy and the flexibility to compute an over-determined least-squares solution, our system is capable of tracking up to six satellites simultaneously. This is accomplished with six GPS *channels*, where each channel is responsible for processing the signal from one particular satellite. The six channels have independent hardware for the majority of their operations, except for some complex math operations, which are handled by shared units.

A comprehensive description of GPS baseband processor architectures can be found in [2], [18], [10]. In what follows, we summarize the choices we have made in our implementation of a GPS baseband processor.

3.1 Acquisition

Before a receiver can begin to track a satellite, it needs to know which satellite to track, an estimate of the Doppler frequency, and the code offset of the signal for that particular satellite. Therefore, during signal acquisition, a typical receiver searches the expected Doppler frequency space and code offset space of candidate satellites.

In the most general case, each satellite is tested against a sets of candidate Doppler frequencies, $\hat{f}_D$, code replicas, X_P. and code offsets, $\hat{\tau}$. These three sets form the candidate hypothesis set $\hat{f}_D \times X_P \times \hat{\tau}$, each of whose elements contains enough information to potentially

recreate the incoming signal. Typically, these hypotheses are tested by correlating them against a received signal over an entire code period, corresponding to 1ms of data, or N front end samples, given that the front end ADC is sampling at a rate of N KHz. To compensate for carrier phase shift, the correlation is done over both *in-phase* and *quadrature* recreated signals, I, and Q, which are exactly 90 degrees offset from each other. Denoting the k^{th} sample from the RF front end by $r_{L1}(t_k)$, and the intermediate frequency to which the satellite signal has been mixed down by the RF front end by f_{IF}, we can write:

$$I = \sum_{k=1}^{N} r_{L1}\left(t_k\right) X_P\left(t_k - \hat{\tau}\right) \cos\left\{2\pi\left(f_{IF} + \hat{f}_D\right) t_k\right\} \tag{3.1}$$

$$Q = \sum_{k=1}^{N} r_{L1}\left(t_k\right) X_P\left(t_k - \hat{\tau}\right) \sin\left\{2\pi\left(f_{IF} + \hat{f}_D\right) t_k\right\} \tag{3.2}$$

If the correlation power, I^2+Q^2, exceeds a threshold, then we have validated a $(\hat{f}_D, X_P, \hat{\tau})$ hypothesis and accept this as an acquisition result for one satellite. There may be more than one hypothesis that exceeds this threshold; in our system we acquire by sequentially testing $\hat{\tau}$ code offsets and selecting the first code offset that exceeds the threshold. Our system does not scan different Doppler frequencies, nor does it scan through the code identifiers, instead requiring that this information be loaded from an external source. This source may be a DSP using an FFT to rapidly search for Doppler candidates, or almanac data being used in conjunction with rough time and position data to estimate SV information. Delay between the off-chip system finding the Doppler frequency and the system concluding the acquisition phase may introduce additional inaccuracy in the initial Doppler estimate.

A rough block diagram of the six-channel system used to do the correlation is shown in Figure 3.1. In each channel, two 32-bit numerically controlled oscillators (NCO) are driven by the data-flow from the front end. The code NCO controls the chipping rate of the code replica generator. Every time the code NCO overflows, a new code replica is generated and every 1023^{rd} overflow of the code NCO marks the end of a 1ms code period, forming our

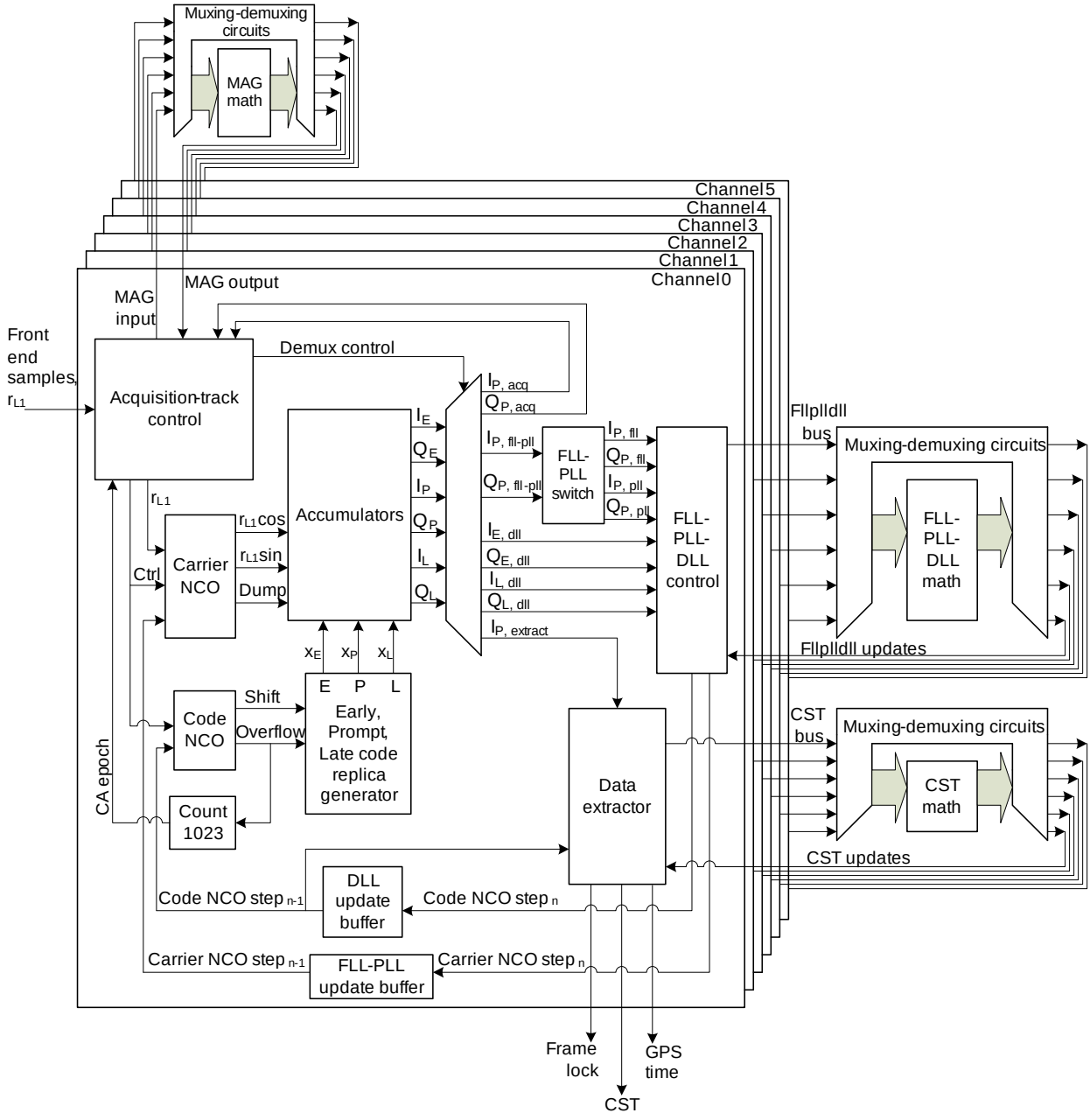

Figure 3.1: System block diagram

accumulation interval. The carrier NCO consumes a data bit whenever one is available, and correlates this against replicated sinusoids to create a 3-bit output for both the in-phase and quadrature components. Multiplied together, these signals form the correlator summands from (3.1) and (3.2).

3.2 Tracking

After acquisition, the receiver channel has enough knowledge of the code offset and Doppler frequency to roughly align its code and carrier replicas to the received signal. The receiver channel then enters tracking mode to continuously track deviations in code offset and Doppler frequency.

Besides the prompt correlators that produce the prompt in-phase accumulation in (3.1) and quadrature accumulation in (3.2), each channel uses early correlators with the replica Gold code advanced by a half chip to produce the early in-phase and quadrature accumulations, and likewise, late correlators with the replica Gold code delayed by a half chip to produce the late in-phase and quadrature accumulations. These early and late accumulations are needed to implement a Delay Locked Loop (DLL) which is used to update the chipping rate for the channel's code NCO to correct for misalignments in phase between the replica Gold code and the received Gold code.

Other than tracking the code phase, the system uses an Frequency Locked Loop (FLL) and a Phase Locked Loop (PLL) to track the Doppler frequency. The FLL is more robust to noise, has better dynamic performance, and has a wider pull-in range than the PLL but its measurements less precise. Hence, the FLL is only used to obtain a more accurate estimate of the Doppler frequency immediately after acquisition before handing the task over to the PLL. This transition from FLL tracking to PLL tracking happens after a programmable delay (400ms was typically used in testing).

The PLL locks the phase of the in-phase portion of the carrier replica to the incoming signal. It is important to realize that all DLL, FLL and PLL tracking loops are only called into action at the end of an accumulation period which occurs once every 1ms in each channel. More details on our tracking loops can be found in Appendix A of [26], where this work was originally published.

3.3 Data Extraction

The GPS satellite's signal contains a 50bps navigation data stream. This data stream is modulated with the satellite's Gold code which has a period of 1ms. As a result, 20 periods of the satellite's Gold code contain information for the same data bit. Since a PLL is used for tracking, the accumulation power is concentrated in the in-phase accumulation. Hence, at the end of a 1ms accumulation period, we can extract one raw data bit from just the sign of the prompt in-phase accumulation.

To remove the redundant data bits, we down-sample the raw data bits by 20 to 1. From the down-sampled data, the system attempts to lock onto the GPS message frame. Each 300-bit subframe begins with the preamble sequence of "10001011", so the system initially simply searches for this pattern.

However, as our PLL discriminator has a 180-degree phase ambiguity, the extracted data bits may be inverted. Thus, a search for both the normal and inverted preamble bit sequence is required. To ensure that the detected bit sequence is indeed the preamble bits instead of a string of navigation data message bits that happens to match the preamble bit sequence, the system extracts the GPS satellite time information and checks that it correctly increments from one subframe to the next before announcing frame lock.

Our system uses a code phase accumulator for each channel, hereafter called Code Start Times (CST), as time stamps in units of sample counts to mark the start of the C/A Gold code period. Every time the DLL updates (approximately 1ms), the CST is incremented by the number of samples in that approximate 1ms time lapse. This number does not have to be an integer, and is instead calculated by the present value of the Code NCO step, allowing for sub-sample resolution. At an interval of 1s, determined by counting 1000 raw data bits from the start of a subframe, each channel reports its frame lock status, GPS satellite time and its corresponding 64-bit CST time stamp.

With CST stamps from the same satellite time from at least four frame locked channels,

an external processor can compute the navigation solution. First, relative pseudoranges are computed from the CST values of each channel using the first channel as the reference [2]. From the relative pseudoranges and corresponding GPS times, we can then compute the X, Y and Z receiver position and receiver clock error [2], [18].

Typical pseudorange measurement rate is 1Hz, which is supported by the system described in this thesis. However, it could be possible to report pseudoranges at a much higher rate by simply making the CST time stamps available at the end of each accumulation interval (once every millisecond). This would add significant communication overhead, but would otherwise not affect the system.

CHAPTER 4

ASYNCHRONOUS DESIGN

In this section, we will give an overview of how the system described in this thesis benefited from asynchronous design methodologies.

4.1 Architecture

Figure 4.1 shows the high-level block diagram of the asynchronous GPS; a simplified version of Figure 3.1. A scan system (further discussed in Section 5.1) loads in required configuration data for the correlators and math blocks, and allows for output scanning from the Preamble/Time Detection unit, which also generates the output. There are six copies of the correlator, downsampler, and preamble/time detection systems to support tracking six SVs simultaneously.

Figure 4.2 shows the high-level block diagram of each correlator. These correlators make up the portion of the GPS that is running at the highest frequency and consume the majority of the active power. Data comes in from the environment and passes through the input buffering and acquisition control block. This stage buffers the input to allow for delay variations, and which controls data flow for switching between acquisition mode and tracking mode. The LFSR here implements the C/A code generator shown in Figure 2.4, augmented with control inputs to enable reproducing all possible code phase assignments, $\hat{\tau}$, shown in Table 2.2. The Carrier and Code Numerically Controlled Oscillators (NCOs) are counters, described in detail in Section 4.2.2, generate a reproduction of the IF carrier signal and control signals for running the LFSR, respectively.

These signals are combined with buffered versions of the LFSR Gold code output (creating the early, prompt, and late versions) and accumulated in the accumulators. For each of the six channels, there are six accumulators, an in-phase prompt accumulator, an in-phase early accumulator, an in-phase late accumulator, and their quadrature versions. The output of

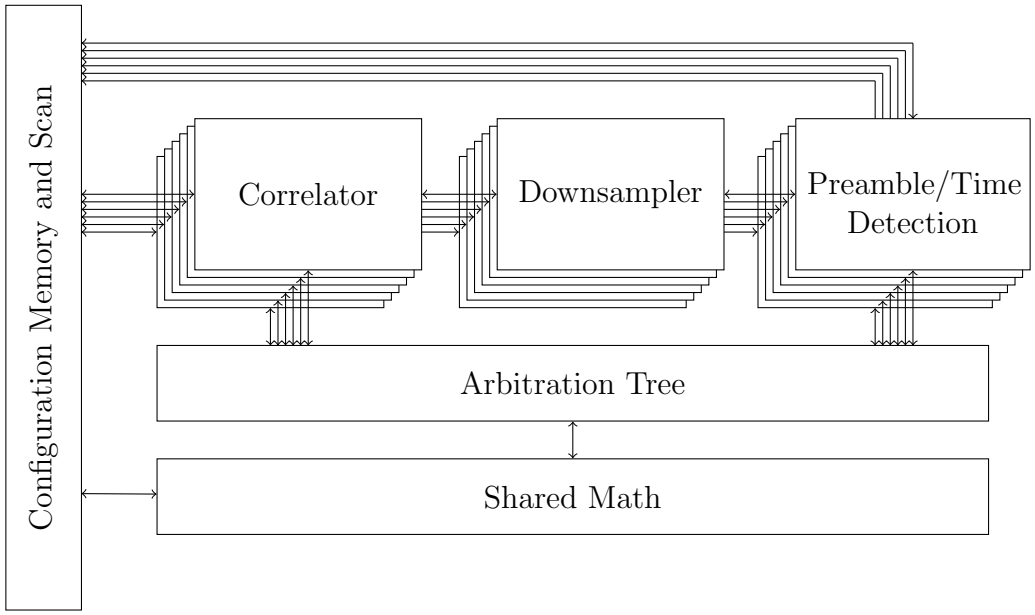

Figure 4.1: High Level Block Diagram of Asynchronous GPS

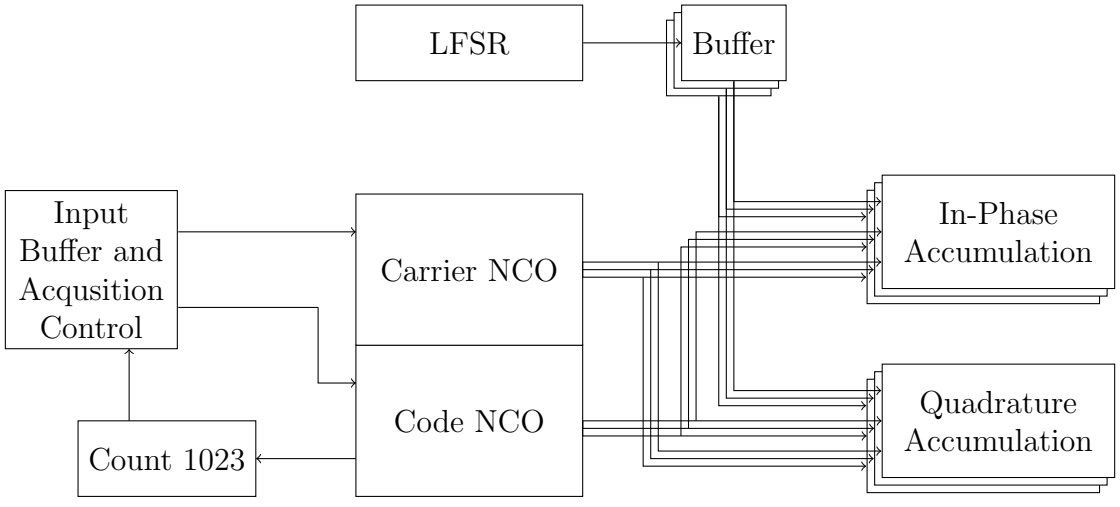

Figure 4.2: Block Diagram Showing Asynchronous GPS Correlation Sub-Block

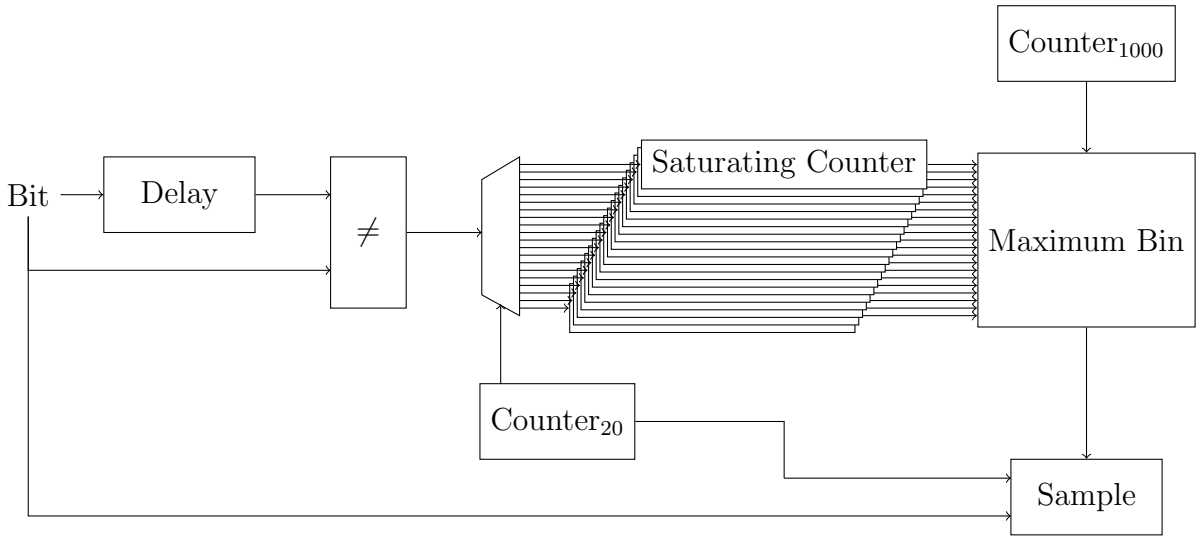

Figure 4.3: Block Diagram Showing Asynchronous GPS Downsample Sub-Block

the in-phase prompt accumulator is the bit for the accumulation window, and the other accumulated values are used to run the Frequency Locked Loop (FLL), Phase Locked Loop (PLL) and Delay Locked Loop (DLL), which make up the majority of the shared math block transistors. The output of these go to the Carrier NCO (FLL and PLL) and the Code NCO (DLL) to maintain their match with the transmitted signals. The DLL output is also used to create the Code Start Time (CST) value, which is used for fine-grained (sub microsecond time tracking).

The downsampler, shown in Figure 4.3 takes the 20 repetitions of each bit, coming in at a rate of 1 kHz, and downsamples them to the actual data rate of 50 bps. It works by detecting when the incoming bit differs from the previous bit, and incrementing the associated saturating counter, which counts up to eight, after which it is considered 'saturated'. After 1000 bits have been processed, the maximum bin block checks to see if one (and only one) of the saturating counters has saturated. If it has, that position is the location of the bit toggle, and this is sent to the "sample" block, to be used to downsample the bit and pass it onto the preamble extractor and time detection unit.

The preamble detection and time extraction unit's high-level block diagram is shown

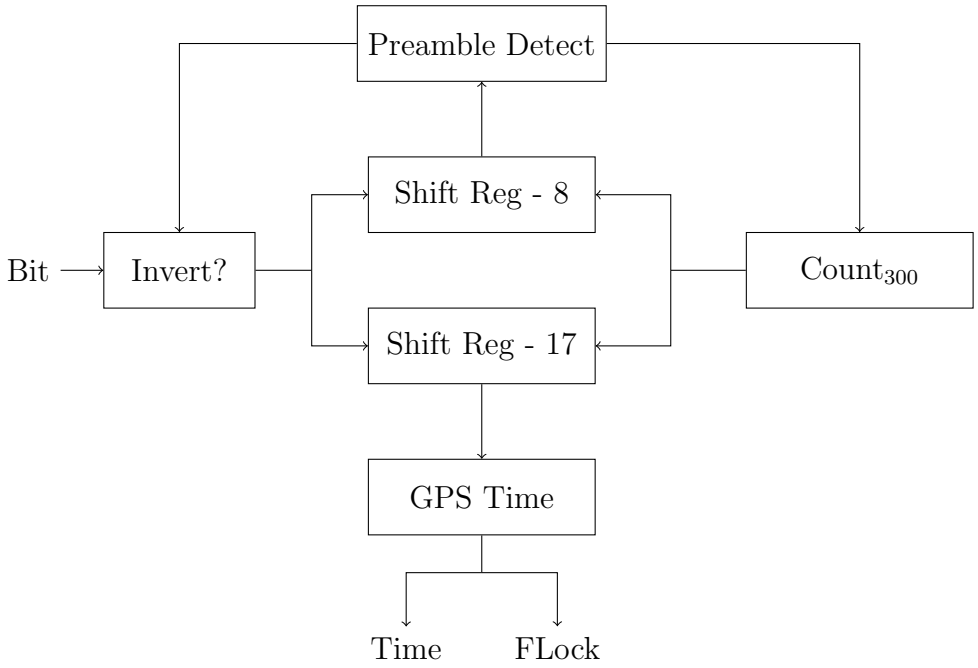

Figure 4.4: Block Diagram Showing Asynchronous GPS Preamble and Time Extraction Sub-Block

in Figure 4.4. This unit first searches for the preamble, which will either be 10001011 or 01110100, due to a 180 degree ambiguity in the phase-locked-loop, and then initializes the 300-count counter, and starts to search for the GPS time, which is known to occur a specific number of bits after the preamble is detected. After the first GPS time is captured, a the search for a second begins. If the second time is exactly one greater than the first, the Frame Lock signal is brought high, and the system begins to output GPS time and CST values once per a second (note that this is six times the rate that the GPS time is actually sent by the satellites, so involves straightforward interpolation of GPS times). If the second time does not match the first, or the system encounters at a later time two sequential GPS times that do not correspond to a single increment, the preamble search begins again.

Note that in the above block diagrams, we did not go into the full detail of each implementation, and we do not show all of the inputs and outputs. This was done to maintain clarity of information, and to focus on the most important parts of each system. Parties interested in the full implementation details should contact the author.

4.2 Subsystem Examples

In this section, the implementation of a selection of the subsystems is discussed. For each subsystem, a description of how it operates is given, and then the procedure of its design and implementation is discussed. The subsystems chosen for this section were chosen because they either give interesting insight into how the asynchronous design enables power saving transformations, or because decisions made during their design led to interesting results.

4.2.1 Adder/Incrementer Accumulator

Accumulators operate at the RF front end sampling frequency and because of that are one of the more power-hungry components in the whole system. Moreover, there are six accumulators per channel to produce early, prompt and late pairs of in-phase and quadrature accumulations.

The 3-bit accumulator inputs are represented by the summands in Equations (3.1) and (3.2) and their early and late variants are accumulated over one accumulation period to produce 16-bit sum outputs. It follows from Equations (3.1) and (3.2) that when the system is tracking well, the iterative cross-correlation of the replica and received signals will tend towards $\pm\infty$, depending on the sign of the encoded navigation data bit. However, for a 5.714 MhZ sampling frequency and a 1 ms accumulation period, they are only accumulating a maximum of 5715 3-bit two's compliment samples, limiting our range to ± 17145, and can therefore be contained entirely in 15 bits. A higher sampling frequency or longer accumulation period would require a larger accumulator, and an analysis for reachable accumulation levels (as the system is accumulating sinusoids, and will never see a situation where all of the accumulation inputs are maximal) might enable a smaller accumulator. We chose to use a sixteen bit accumulator as it will accommodate the worst-case aberrant behavior for a 5.714 MHz sampling frequency while still enabling typical-case accumulation for higher sampling frequencies.

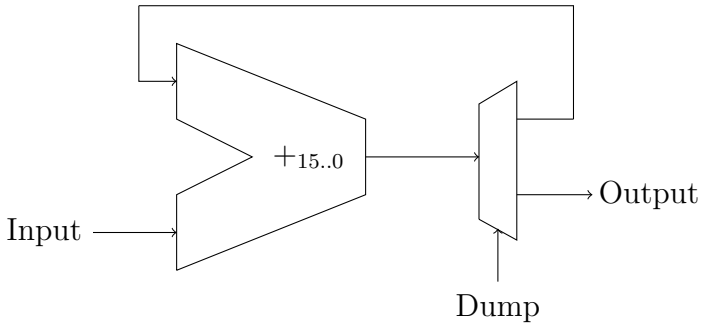

Figure 4.5: Block Diagram Showing Original Accumulator

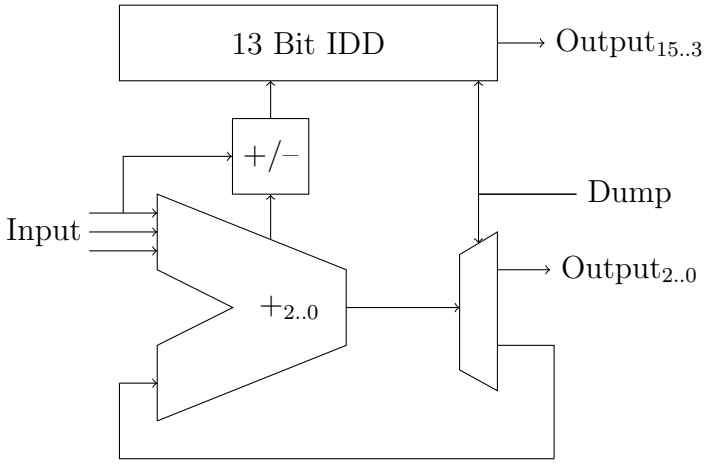

Figure 4.6: Block Diagram Showing Final Accumulator

The simplest accumulator design is shown in Figure 4.5, and in fact was the first system that we implemented. However, as the 3-bit summand is iteratively added to the 16-bit sum, the bits of higher significance do not switch frequently. Based on this observation, we introduce an increment-decrement-based accumulator design to reduce transistor switching activity and hence the dynamic power consumption of this critical module.

We took the original 3-bit accumulator and coupled it with a 13 bit increment-decrement-dump unit (IDD). Only a minimal number of low order bits (three, matched to the input width) are required to cycle in and out of the adder in the standard 3-bit accumulator. The rest of the higher order bits are handled by the IDD. This design is shown in Figure 4.6.

In this system, an encoder issues the command to increment or decrement based on the polarity of the input summand and the carry out from the adder. If the input to the

Table 4.1: Comparison Between Naïve and IDD-Based Accumulator Design

	Naïve	IDD
Transistors	7,011	5,995
Energy Per Accumulation (Aligned)	7.12 pJ	2.58 pJ
Energy Per Accumulation (Misaligned)	7.02 pJ	4.36 pJ

accumulator is positive, the encoder issues an increment command if there is a carry out from the adder.

As the negative of a 2's complement number a ($a > 0$) represented with n bits is written as: $-a = 2^n - a$, where a "borrow" is obtained from the virtual $(n+1)^{th}$ bit. Therefore, if the input to the accumulator is negative, the encoder issues a decrement command if there is no carry out from the standard accumulator as it would otherwise cancel out the "borrow" loaned from the higher order bits in the IDD. If neither of these cases hold, the encoder issues no command and triggers no transistor switching activity at all in the entire IDD.

We designed the 13-bit IDD with 13 cells, each of which implements a need-based command propagation structure. If the IDD receives a command to increment, the Least Significant Bit (LSB) cell will increment its own value and will only propagate the increment command to the next higher-order cell if the LSB cell has a carry out; otherwise the higher-order cells experience no switching activity at all. Other than the Most Significant Bit (MSB) cell which does not need to propagate any commands further, all higher-order cells have the same need-based command propagation behavior as the LSB cell. Likewise, each IDD cell only propagates the decrement command if it needs to "borrow". The dump command which the accumulator receives once every 1ms is propagated throughout the IDD from the LSB to the MSB cell. The full accumulation result is just a concatenation of the result from the standard 3-bit accumulator and the result from the 13-bit IDD.

As the IDD is simpler than a full adder with completion detection, this change reduced both transistors and energy usage, as can be seen in Table 4.1. For this table, and all subsequent tables, power and energy numbers were obtained from HSIM simulations performed at

25 C and 1 V, assuming IBM's 90 nm low power process, and a 5.714 MHz front-end sampling frequency. All synthesis from CHP was done by hand, using Martin's synthesis method [16]. Transistors were sized using a combination of automated tools (discussed further in Section 5.2) and hand optimization.

Also note that in Table 4.1, two different numbers are given for energy per accumulation. This is done to show one of the interesting properties of the IDD approach: when the accumulation does not trend towards a large number, but instead oscillates around 0, it will use almost twice the amount of energy per token as the entire IDD carry chain has to switch to change the sign. The unaligned case occurs in acquisition, causing a slightly higher power consumption during that time. This is also an example of how the QDI asynchronous design is advantageous: the power consumption in the typical case is lower without any additional circuitry to explicitly switch between cases. As the activity does not change between the aligned and misaligned datasets for the naïve accumulator, the energy per accumulation does not change significantly.

These energy numbers are the result of averaging the energy of two hundred accumulation tokens. For the aligned case, these were all maximal tokens (3), and for the misaligned case, it began with a -1 token, and then alternated between maximally positive (3) and maximally negative (-3) tokens. By doing this, the carry chain transitions with every token, resulting in the worst-case behavior. This worst-case behavior is not exactly representative of the behavior during misaligned acquisition, as there may be periods of unidirectional accumulation in a misaligned signal.

It is important to remember that there are six of these accumulators per each of the six channels in the GPS receiver system, so any improvement in the incriminator is amplified by a factor of 36.

4.2.2 Carrier Numerically Controlled Oscillator

The Numerically Controlled Oscillators (NCOs) also operate at the front end sampling frequency and take up a large portion of the system power. For each channel, there are two different NCOs, the Code NCO, which controls the rate at which the LFSR replicates the Gold code, and the Carrier NCO, which controls the rate at which a replicated sine and cosine are produced, replicating the IF carrier signal.

The outputs of the Code and Carrier NCOs are represented in Equations (3.1) and (3.2) as X_p and $\left(f_{IF} + \hat{f}_D\right)$, respectively. The frequency of these signals changes over time, based on the results of the DLL or PLL/FLL respectively, (which themselves are reacting to changes in the satellite Doppler shift and other signal-modifying effects). The computation of the DLL and PLL/FLL are relatively slow operations, particularly if we constrain them to low-energy implementations, and so instead of using the step value calculated from the previous accumulation, the NCOs use the step value calculated from two iterations past. This introduces a slight error in the final navigation results, but through simulation, we found that this error was limited to 20 cm, and was acceptable giving the savings in energy and design complexity. The NCOs were implemented with 32 bit accumulators, to match the NCOs from the Cornell GPS Laboratory's GPS simulator.

For the purposes of this example, we will limit the remainder of the discussion here to the Carrier NCO. Many of the same things are true about the Code NCO.

The Carrier NCO is represented with the CHP (an informal description is given in Appendix A) shown in Figure 4.7. There are three possible operations for the Carrier NCO. The first is to increment the stored value with the stored step value, and then output the sine and cosine values to the accumulators, while simultaneously forwarding the control value down to the accumulators. This corresponds to the accumulators and NCOs incrementing their values, and is the typical operation. The second is to simply obtain a new step value, and forward this control to the accumulator, which will cause the accumulator to dump

```
*[CTRL?c;
   [ c = 0 ⟶
       nco := nco + step,
       ACCCUM_C!0,
       DATA?data;
       SIN!(data × sin(nco[31..29]),
       COS!(data × cos(nco[31..29])
    [] c = 1 ⟶
       ACCUM_C!1,
       STEP?step
    [] c = 2 ⟶
       nco := 1
 ]]
```

Figure 4.7: CHP for the Carrier NCO

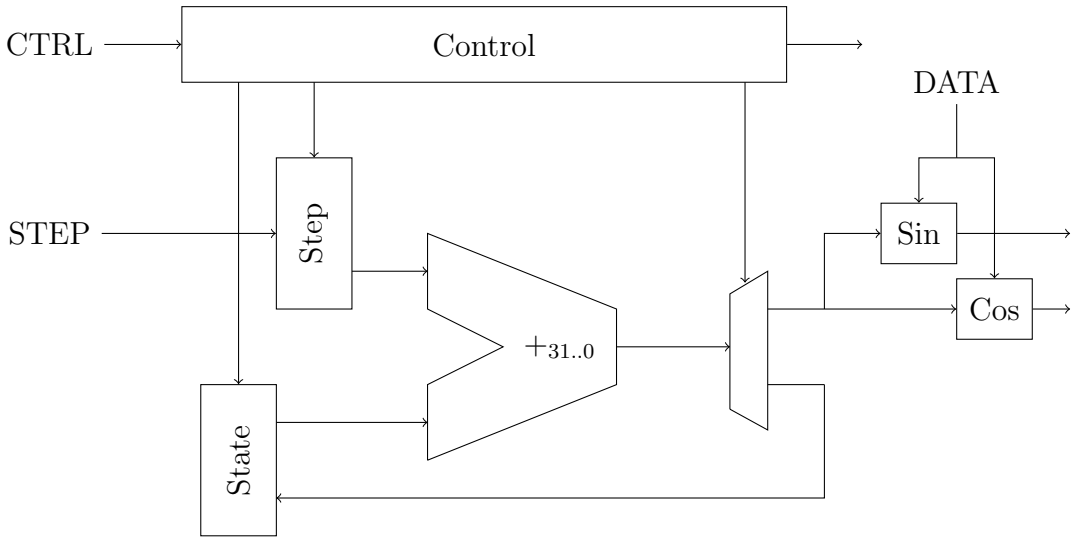

Figure 4.8: Block Diagram Showing Original Carrier NCO

its output, which occurs once every millisecond. The last is a reset command, used when transitioning from the acquisition stage.

The first implementation of this system is shown in Figure 4.8. This implementation uses a control/data decomposition style, where a separate control block controls the flow of information through a function-block style adder. The sine and cosine functions were implemented as 4-bit lookup tables, using the three most significant bits of the NCO combined

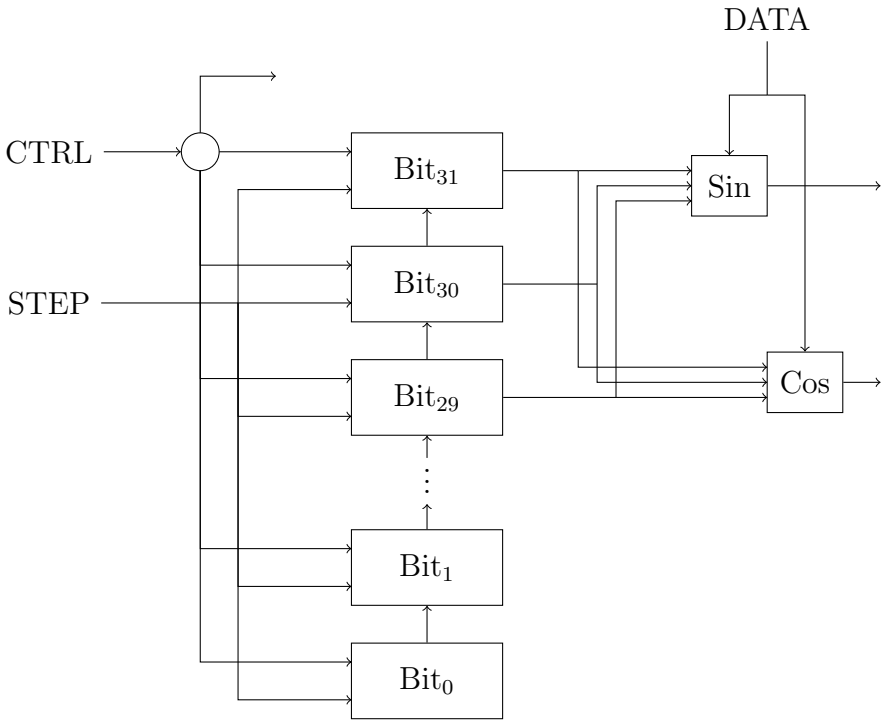

Figure 4.9: Block Diagram Showing Final Carrier NCO

with the single bit of the data. This was found to result in a simpler design than a 3-bit lookup table and conditional negator. This style has a clear correspondence to the original CHP, however, requires completion detection in many places. As the adder, registers, and deterministic split (demultiplexer) are all 32-bit, this adds in quite a bit of transistor and energy overhead.

The second revision, shown in Figure 4.9 shows the implementation in a bitwise-projection style. Each one of the individual bits has been separated out as its own process, handling the storage of its associated step value, its current state, and its control needs. The control value is copied out to all of the different bits (and conditionally to the accumulators), while the step value is fanned out to the bits.

This organization also enabled a second optimization, in that now that each bit in the adder is communicating directly, the carry bit can be produced as soon as enough information is available, instead of waiting for all of the inputs to arrive. While this does not improve the

Table 4.2: Comparison Between Carrier NCO Designs

	Control-Data	Bitwise Projection
Transistors	14,141	11,904
Energy Per Token	14.4884 pJ	13.903 pJ

worst case performance, it can give large average-case gains, as the carry chain is no longer always 32 bits long, as it is in a simple delay-insensitive function-block adder. These gains were found to more than compensate for the additional overhead of communication between the blocks.

There are four different kinds of bit blocks used in this design, which are all represented in Figure 4.9. The first is the least significant bit, Bit_0, which only requires a half-adder to do its addition. The second is the common case, Bit_1, which has both a carry-in and carry-out, but does not need to communicate its sum output. The third are the two penultimately significant bits, Bit_{29} and Bit_{30}, which have both carry-out and sum out outputs, and the last was the most significant bit, Bit_{31}, which does not have a carry-out output. The sine and cosine block are identical to the previous version.

Table 4.2 shows the difference in transistors and power for the two different implementations. These simulations were run at 5.714 MHz, and represent the average and standard deviation of the energy of 100 tokens. Both the designs were run using the same sequence of step and control values, which were obtained from a run on data from a real GPS frontend. More details on the simulation environment can be found in Section 6.2.

Going from the control-data decomposition style to the bitwise projection saves 16% on transistors, but only 4% on energy per token. This disparity makes intuitive sense as even though the number of transistors has decreased, the same function is being performed, and the core power consumer, the addition, has remained the same. Also, while the bitwise decomposition removes the need for global completion detection, it adds on local control overhead, making each bit more complex. This was found to give a significant increase in

maximum throughput, though that is not reflected in the simulation, as both designs meet the minimum required throughput of 5.714 MHz.

4.2.3 Downsampling

The downsampler does not consume a large amount of power, as it runs at the raw data frequency of 1 KHz. However, the algorithm being run may require a large amount of storage, and the first implementation used an unacceptably large number of transistors. This section discusses some of the algorithmic improvements that were used to decrease this transistor count, and thus the area overhead.

Figure 4.10 is a CHP representation of the algorithm that the downsampler is implementing. Functionally, the purpose of the downsampler is to take the 20 repetitions of each bit and transform them into a single bit. For GPS applications, knowing where in the sequence the bit transition occurs is very important for position accuracy, as a single-bit error corresponds to a timing error of 1ms, or a position error of approximately 300 kilometers, well outside of the acceptable bounds. This requires a more complicated downsampler than the typical 'synchronized sampling' method.

The downsampler maintains 20 bins, each representing 20 possible 'locations' for the bit transition (where bin 0 represents the 0th bit received by the downsampler, 1 the second, and so on, modulo 20). Every 1000th raw bit received, the downsampler checks to see if one of these bins is larger than all of the others and exceeds a threshold. This threshold is in place for the case of a long sequence without any transitions, but with a small number of 'glitches', where one of these may accidentally be accepted as the bit transition. A threshold of ten was chosen as it is not too large to prevent a small number of real transitions from giving an accepted lock while still preventing glitches from gaining lock. If these checks are passed, the downsampler accepts this bin as the transition location, and begins to send bits to the preamble detection and time extraction stage.

```
*[ RAW_BIT?raw_bit
  [ raw_bit != old_bit  ⟶
      bin[bit_count%20]  =  bin[bit_count%20]  +  1
  ];
  [ bit_count  =  1000  ∧  ¬bitlock  ⟶
      i  :=  0,  max  =  0,  maxbin  :=  0
      *[ i  <  20  ⟶
          [ bin[i]  >  max  ⟶  max  =  bin[i],  maxbin  =  i
          [] else  ⟶  skip
          ];
          bin[i]  :=  0
      ];
      [ max  >  10  ⟶  bitlock↑ ]
  ];
  [ bit_count  =  maxbin  ∧  bitlock  ⟶  BIT!raw_bit
    PREAMBLE_FOUND?p_f;
    [ p_f  ⟶  sample_time  =  (bit_count − 160) ∧ 1000,  found↑
    [] else  ⟶  skip
    ]
  ];
  [  bit_count  =  sample_time  ∧  found  ⟶  INC_TIME!1
   [] found  ⟶  INC_TIME!0
   [] else  ⟶  skip
  ];
  old_bit  :=  raw_bit,
  bit_count  :=  (bit_count  +  1)%1000
]
```

Figure 4.10: CHP for the Downsample

Once the preamble has been discovered, the position of the beginning of the frame (8 bits, or 160 raw bits) is saved. When the 1000-count reaches this position again, a signal is sent to the next section to increment the currently stored time value and to produce an output. This gives us an output every one second even though a new time stamp is only sent from the GPS every six seconds. This was done to maintain compatibility with standard GPS receivers, which also interpolate time at one-second increments.

Figure 4.11 shows the implementation of this downsampler, though it does not contain the blocks for generating the *INC_TIME* signal, as they clutter the diagram without re-

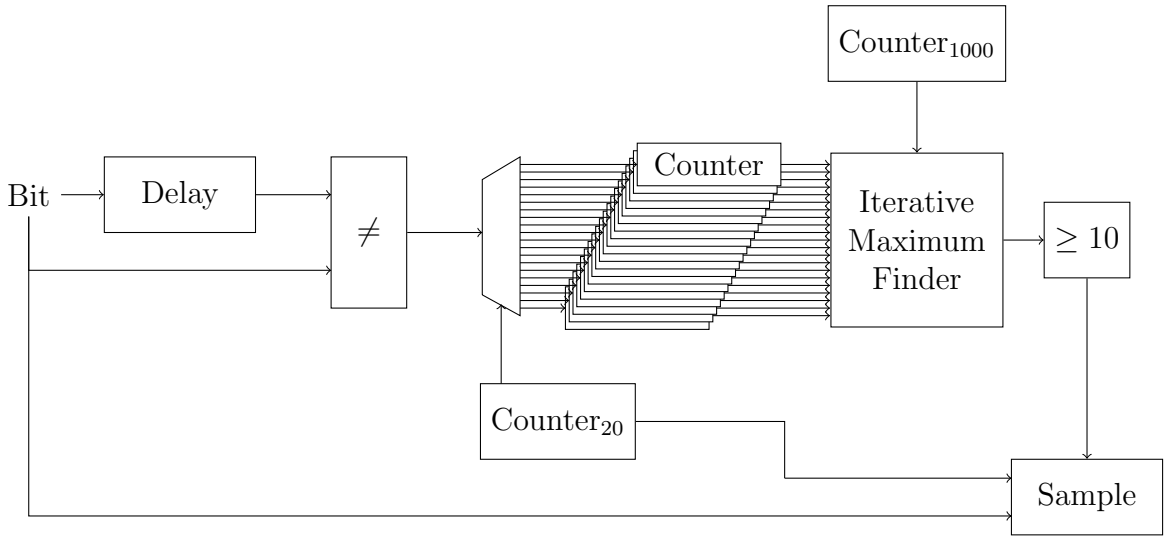

Figure 4.11: Block Diagram Showing Original Asynchronous GPS Downsample Sub-Block

vealing any interesting information. These blocks connect to the 1000-cycle counter, and communicate with the preamble-time detection unit in a relatively straightforward fashion. Each of the bin counters in this diagram is six bits long, as each must be able to handle the maximum of 50 transitions in one 1000 bit period. The maximizer was implemented with an iterative design, to cut down on transistor cost in exchange for a significant speed penalty. Overall, this design uses a large number of transistors, over 100,000, and a better design was needed.

This better design came in the form of using 4-stage saturating counters, which would enter a 'saturated' state after being incremented eight times. This design was shown in Figure 4.3 as the block diagram for the downsample unit. Using the saturating counters required some algorithmic changes, so that the used downsampler does not exactly implement the algorithm above.

Instead, when 1000 raw bits have been seen, all of the counters are checked to see if any of them have saturated. If one, and only one has saturated, it is accepted as the transition bin. If more than one has saturated, neither is accepted, and another accumulation period begins. This differs from the original algorithm in two major ways. First, the threshold is

Table 4.3: Comparison Between Downsample Designs

	Maximizing	Saturating
Transistors	103,990	45,043

now eight instead of ten, this is simply because a counter that saturates at a power of two is much simpler. Secondly, if two counters are above threshold, neither of them is accepted in the saturating downsampler, while the larger one of them would have been selected in the iterative maximization downsampler.

Through simulation, these differences in behavior were found to not cause any operational differences, except in one case, when the system was incorrectly tracking. In this case, multiple of the saturating counters saturated, and the associated downsampler did not sample the signal, while the maximizing downsampler locked onto a nonsense bin. When the system began to correctly track, the saturating downsampler was able to correctly identify the sampling bin.

Table 4.3 shows the difference in transistor usage by the different downsampling implementations. As they are operating at such a low frequency, and are typically doing very little, energy numbers do not give any interesting points for comparison, and are typically dominated by leakage power. However, the saturating counter method uses fewer than one half of the transistors of the maximizing method while still accomplishing the required operations. Therefore, the saturating downsampler was selected for the final implementation.

4.2.4 Control Signal Generator

In several locations in the GPS, a certain number of tokens must be counted (either to skip them, load them into shift registers, or simply to keep track of position in the computation) before an action occurs. The naïve way of accomplishing this is to keep a counter which continually increments until it reaches some value. However, if that value is not a power of

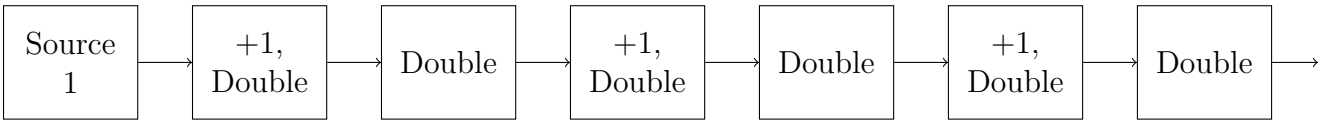

Figure 4.12: Block Diagram Showing Static Control Signal Generator for Forty Two Zeros

```
*[ IN?c;
   [ c = 0  ⟶  OUT!0;  OUT!0
   [] c = 1  ⟶  OUT!1
   ]
]
```

Figure 4.13: CHP for the Control Signal Generator Doubler

```
*[ IN?c;
   [ c = 0  ⟶  OUT!0;  OUT!0
   [] c = 1  ⟶  OUT!0;  OUT!1
   ]
]
```

Figure 4.14: CHP for the Control Signal Generator Plus One Doubler

two, doing this comparison against every value will be costly. For the GPS receiver, a control signal generated was created, which would transmit N number of zeros, followed by a one. There were two designs made, one which was static, and would always send the same number of zeros, and another which was dynamic, sending a different number of zeros dependent on the value of an input. In this section, we describe these two designs, and give comparisons of power and energy for the two implementations.

Figure 4.12 shows a control signal generator that will generate a pattern of forty-two zeros followed by a one. It is made of two components, whose CHP can be seen in Figures 4.13 and 4.14. The when the Doubler receives a zero, it will output two zeros, and when it receives an one, it will output an one. When the Plus One Doubler receives a zero, it also outputs two zeros, but when it receives a one, it outputs a zero followed by a one. To build up a static control signal generator we first convert the signal to binary, for forty-two this is 0b101010. Starting with the most significant bits chain together a Plus One Doubler in the position of every one and a Doubler in the position of every zero. Connecting the most significant bit to a one source completes the circuit. As is expected, any preceding zeros in the binary representation do not have an effect on the number of zeros output from the Control Signal Generator, as the source alone will not produce any zeros to be doubled.

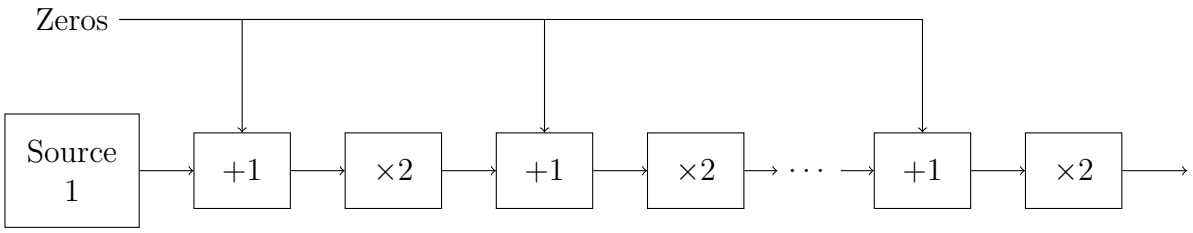

Figure 4.15: Block Diagram Showing Dynamic Control Signal Generator

Table 4.4: Comparison Between Control Signal Generator Designs

	Static	Dynamic
Transistors	829	1,913
Energy Per Sequence	13.811 pJ	14.162 pJ

Table 4.4 was generated through SPICE simulations as described above. The energy result is the average of ten full sequences of forty two zeros followed by a one. Unsurprisingly, the dynamic generator uses more than twice the number of transistors that the static generator uses. It requires completion detection for the input control value, as well as additional logic in the plus one detectors to make them conditional. However, it does not use a significantly larger amount of energy over the course of a full sequence generation, as this additional circuitry is only rarely enabled.

CHAPTER 5

IMPLEMENTATION

This section will cover some of the challenges that were overcome in the implementation of the GPS in IBM's 90 nanometer low-power process, commonly known as IBM9lp. It will go through some of the specific challenges of implementation, tools used or implemented to make the procedure simpler, as well as justifications for some of the design decisions made.

5.1 Configuration Memories

When designing the GPS channels we decided ahead of time to avoid hard coding as many constants as possible to preserve generality. However, this creates the need to load that data into the system though a separate step before processing can begin. In this subsection, we will describe the method that we chose to use, the system design, and the analog verification that was used to verify functionality.

5.1.1 Input/Output

The system used is similar to an SRAM decoder, where a token is used to address each dimension. A simplified two-dimensional diagram of this can be seen in Figure 5.1. In this system, two non-overlapping clock signals are used to move a token horizontally and vertically until the bit of interest is addressed. This bit can then have its current value overridden by driving the `_wt` and `_wf` data lines, or it can have its current value read through the `Out` data line. These values are propagated through buffers in the Horizontal shift registers, as can be seen in the circuit diagram in Figure 5.2

Note that in the Horizontal Shift registers, there is a `pe`, or Precharge Enable signal. This signal is used to precharge the bitlines between accesses. For the circuits implemented, this signal was driven by the NAND of the clocks. That means it would be driven only when both of the clocks are disabled. Also note that this circuit will only pull the bitlines

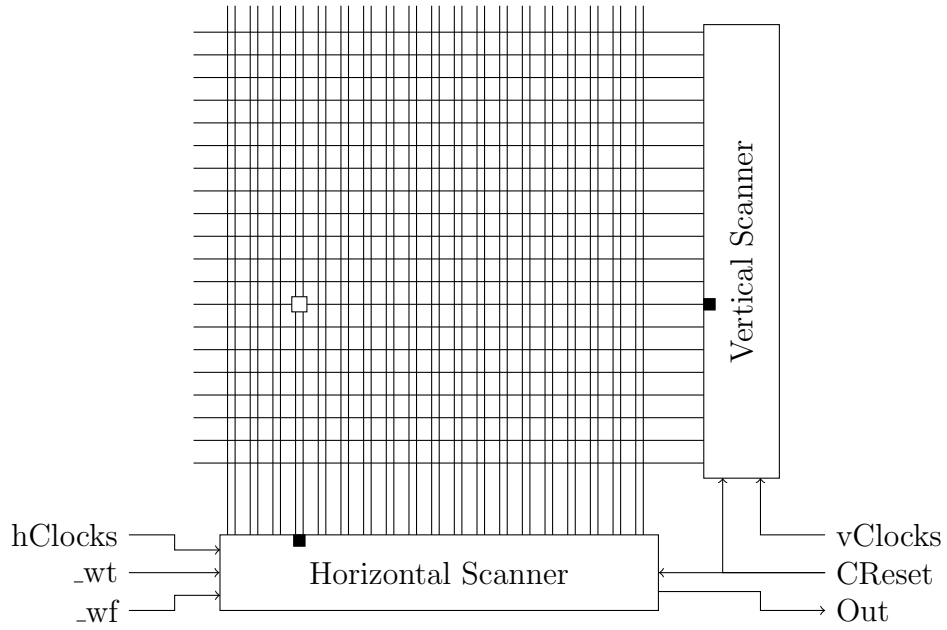

Figure 5.1: Two Dimensional Token Decoder with Token at (3,7)

down when `hClock2` is high, ensuring that it is never the case that the precharge is pulling the bitlines high while the logic is pulling the bitlines low. There is a timing assumption in that guarantee, specifically that the precharge enable line will be discharged before the pulldown begins. This was verified through simulation, and is simple to satisfy as that the precharge enable pull-up logic is only has a single gate latency, while the pulldown logic has a three-gate latency.

The simple two-dimensional decoder works for smaller memory arrays, but a different design is needed to scale to the large amount of signals we needed to route, particularly while keeping the amount of capacitance on the bitlines low. Enter the three dimensional decoder, seen in Figure 5.3, which adds an additional "Zone selector" token buffer, similar to the two dimensional horizontal token buffer. This allowed us to keep the individual bitline wire capacitance very low while still giving a large amount of addressable space and a low pin count. The total final pin count for this decoder was 10: `hClock1`, `hClock2`, `vClock1`, `vClock2`, `zClock1`, `zClock2`, `CReset`, `_wt`, `_wf`, and `Out`, however, it allowed us to address as many different internal bits as needed.

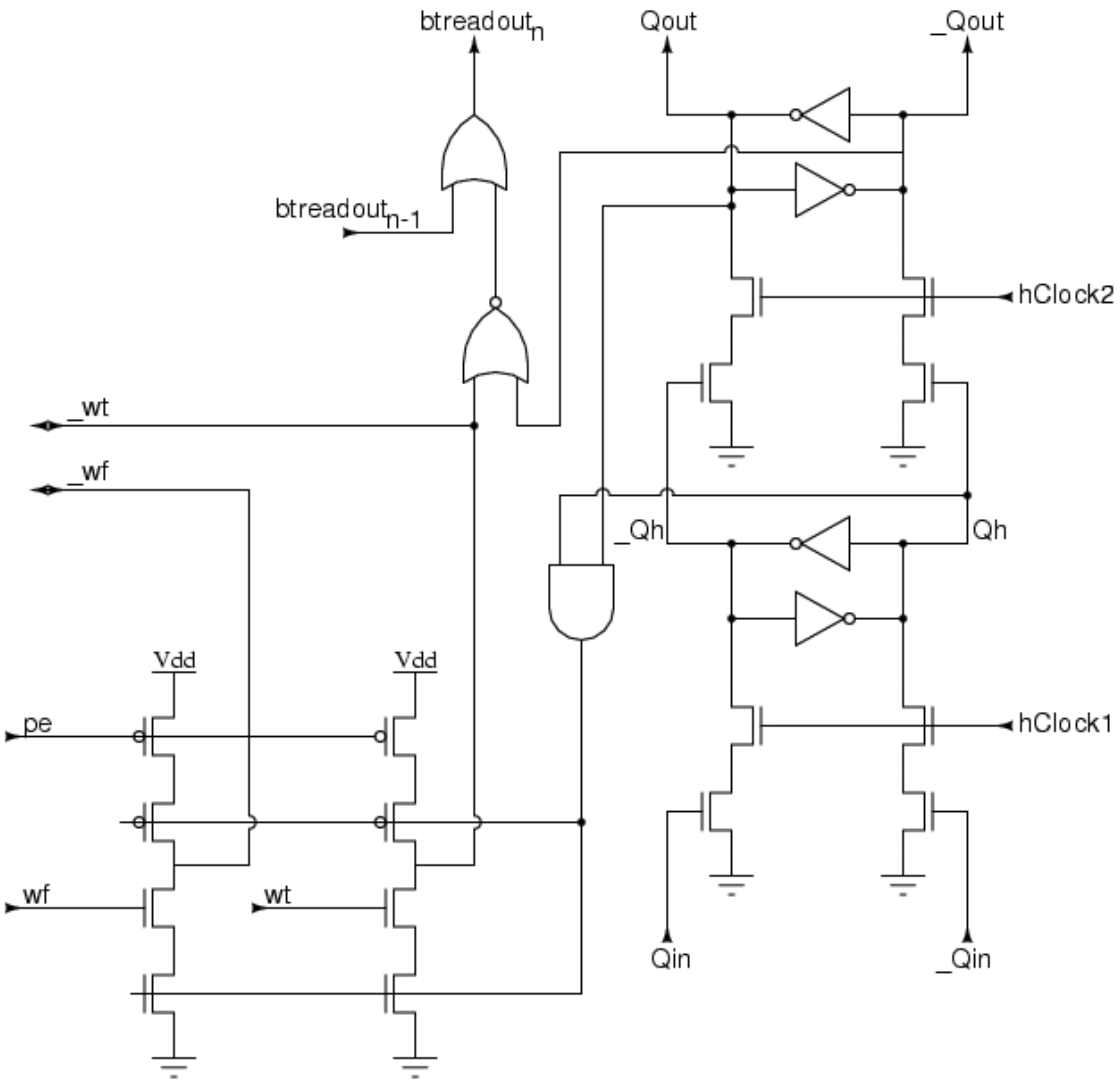

Figure 5.2: Horizontal Token Shifter

5.1.2 SRAMs

Six-transistor SRAMs were used to implement the configuration memory state-holding cells. These were sized and their noise margins were verified using conventional methods [1] [24].

The test bench and results for the hold and read static noise margin can be seen in Figures 5.4 through 5.7. Post-layout extracted measurements of the bit wire capacitances were used to ensure that these margins would not be violated. As these measurements are of the static noise margin, this is conservative. A dynamic noise margin measurement may

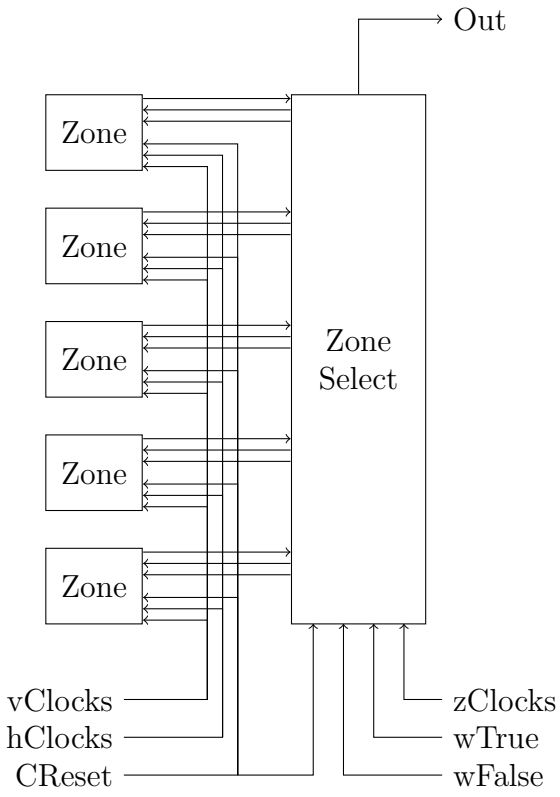

Figure 5.3: Three Dimensional Token Decoder

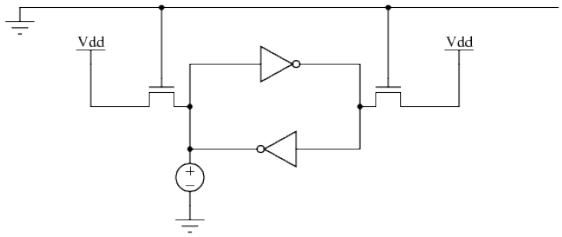

Figure 5.4: Static Noise Margin Hold Test Circuit

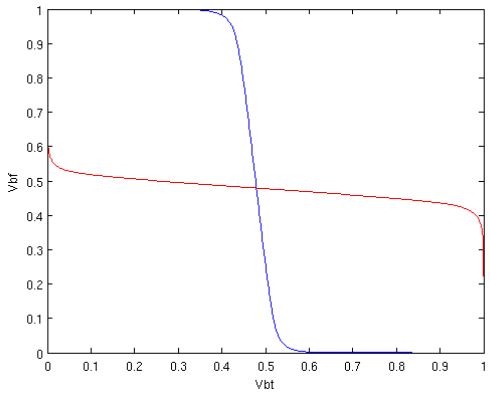

Figure 5.5: Static Noise Margin Hold Test Plot

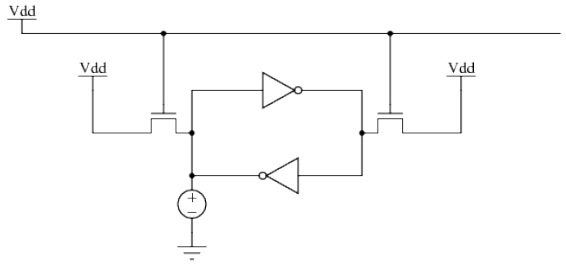

Figure 5.6: Static Noise Margin Read Test Circuit

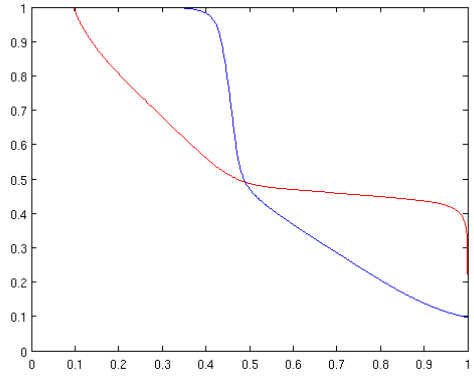

Figure 5.7: Static Noise Margin Read Test Plot

have allowed for longer bit lines and larger zones, but would give less room for error.

5.1.3 GPIO Usage

The configuration memory decoder can also be used for General Purpose Input/Output (GPIO) by using it to address an arbitrary signal, and then reading the value of the Out line after waiting for the value to propagate. To keep these taps from adding significant capacitance to the node of interest, inverting amplifiers were connected between the node and the False output line. These amplifiers were enabled by the bit select lines from the vertical buffers.

For reasonable access times on these GPIOs, the signals of interest should be connected near the end of the horizontal and zone buffers, to prevent them from needing to propagate values a significant distance. Regardless of the placement in the chain, the design is not sufficient for anything approaching rapid access. In the GPS receiver design, this GPIO usage was only added for two purposes: adding additional testing taps, designed to be accessed when the system is stalled and the final output buffers, which are only to be accessed once every second.

5.2 Sizing

After synthesizing circuits through Martin Synthesis, they need to be given exact sizes before they can be physically realized. These sizes can be chosen to meet a variety of metrics, including power, performance, and area. To speed up the sizing procedure, we chose to use a tool developed by Tony Lee at CalTech, known as ERGEN [11]. This tool is based on the Event-Rule (ER) systems [4], which are a system for analyzing the performance of QDI circuits. The tool uses a subgradiant optimization method to minimize the maximum cycle latency, estimated using the Tau delay model.

As this tool had not been used for any purpose in over a decade, there was significant work that went into determining its interface, and in getting it to compile again to make simple operations. The documentation for this interface can now be found on the Cornell AVLSI wiki[1]. Through using this tool, we discovered several limitations which may restrict its practical usability in modern systems. We were able to create scripts to compensate for many of these limitations and utilize it in the sizing of the GPS.

One limitation is that given the Tau model, the optimal transistor sizing for some transistors is near infinite (e.g. the Reset transistors, which are static throughout the operation, but contribute resistance to the gates). Additionally, as event-rule systems give a separate event for each transition, not for each transistor, the system has to be seriously augmented to support PRS sizing, which was done through a series of Python scripts.

A more substantial limitation, and one that must be carefully dealt with, is that the complete ER system must include an environment, and the sizing is only optimal for that specific environment. In sizing the GPS receiver, this was handled by constraining the sizes to maintain certain symmetries, e.g., all bits of an adder must have the same sizing, though this invalidates the optimality claim from ERGEN's resultant sizing. It was also sometimes possible to handle this limitation through a sufficiently general environment, but by being

[1] http://vlsi.csl.cornell.edu/wiki/doku.php?id=designtools:ergen

more general, the ER system created was larger, and the overall system took quite a bit longer to generate and optimize. In the worst case realized, this took several days on a modern machine. Others ran for over a month before being stopped, with no results obtained.

For many of the critical blocks, e.g., the NCOs, the ERGEN sizing was used as an initial guideline for the hand sizing of the blocks. In this process, we would use ERGEN to give a sizing given some environment that was assumed to be sufficiently general, though may not exercise all possible cases, and then would run the system through a SPICE simulation of a different, but still sufficiently common-case environment, and look for slow slew rates, or subsystems consuming disproportionate amounts of energy. These would then be hand-optimized to give performance more in line with what we desired.

Overall, the semi-automated sizing process likely saved us a great deal of time compared to the hand sizing of these circuits.

5.3 Layout

For the system to be physically manufactured, physical placement for each of the transistors and specific routes for each of the wires must be generated. Like in the sizing procedure described above in Section 5.2, we went with a semi-automated approach. The automated portion came from the tools LayoutTK and CellTK, which were being constructed during the layout procedure by Robert Karmazin and Carlos Tadeo Ortega Ortero, respectively. LayoutTK is a Python toolkit for taking in SPICE descriptions of circuits and generating layout. CellTK is a program which will take as an input an Asynchronous Control Toolkit (ACT) Language description of a circuit, break that circuit into smaller pieces (similar to standard cells), has LayoutTK generate the layout for these standard cell, and then pass them into Cadence Encounter (an industrial standard cell place and route tool), which places and routes them.

The reason that this procedure was semi-automated instead of wholly automated was that

LayoutTK was not yet perfect, and cannot create layout for all standard cells, particularly in the case of complex cells. In these cases, standard cell layout was generated through hand placement and routing. LayoutTK is also not always capable of producing layout that satisfies the Design Rule Check (DRC), though these errors are not related to circuit complexity. These cases were identified by the Mentor Calibre DRC tool, and were corrected by hand. Future revisions of the LayoutTK tool may not suffer from these problems.

Another source of errors is that Cadence Encounter does not always successfully synthesize circuits, and needs external guides to complete the layout well. This is further complicated for asynchronous circuits, where the timing driven placement of Cadence Encounter does not apply correctly, and the cycles' timing constraints will not be respected. Cadence Encounter also has known problems where the layout will not be generated in a way that satisfies the DRC in some situations. Similar to the LayoutTK case, these errors were identified by the Mentor Calibre DRC tool, and were corrected by hand.

Despite these problems, the flow was successful, and likely saved us a great deal of time over fully manual hand layout. The layout shown in Figure 5.8 is the GPS system that has been described in this thesis, along with test structures for the NCOs and Accumulators, to allow them to be tested independently of the other systems. In total, there are over 3.2 million transistors on this chip, which was laid out on a 4mm by 5mm silicon rectangle. The distribution of transistors can be seen in Table 5.1. This table does not include the decoder and configuration usage, nor does it include the intermodule buffering, so the floorplan block total does not always match the total of the contained modules.

5.3.1 Floorplanning

Figure 5.9 Shows an approximate representation of the floorplan for the GPS system. Each individual channel was broken into 3 components: `chanCorr` (in blue) contains the accumulators, NCOs, and other high-frequency subsystems, `chanTrack` (purple) contained buffer-

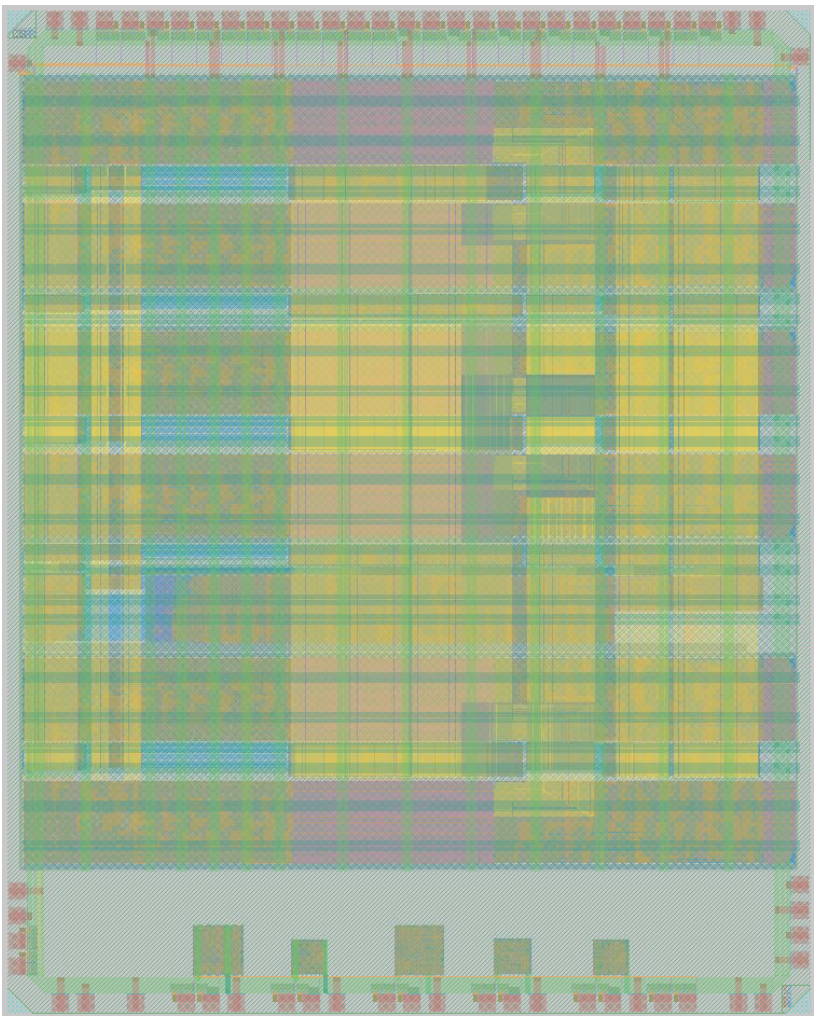

Figure 5.8: Image of the Full Chip Layout

ing for communicating with the shared math blocks, and `chanData` (orange) contained the preamble/time detecting and downsampling circuitry. The arbitration trees (grey) for the shared math blocks were spatially distributed throughout the chip, directing data to and from the shared math blocks (green). The math blocks were laid out in a separate process, mimicking traditional synchronous standard layout. This was possible as they are synchronous synthesized blocks, originally described in Verilog, and accompanied with matched delay lines to allow them to be treated as bundled-data asynchronous systems.

Not shown in Figure 5.9 is the configuration memory decoder chain and buffers. These went along the top of each channel, and were used in buffering the reset signals to ensure

Table 5.1: GPS Transistor Distribution

Subsystem	Module	Transistors
chanCorr		111867
	LFSR and Buffers	5184
	Acquisition	39272
	NCOs	22682
	Accumulators	35970
chanTrack		122726
chanData		122726
	Downsample	45043
	Preamble/Time	77683
Shared Math		566164
	FLLPLLPLL	429068
	WFR	55662
	MAG	8580
	CST	72854
Arbitration		518785

that they were distributed properly as well as buffering the control signals for the decoders. This was done to ensure that the slew rates for all of these signals was acceptably high while not needing excessive buffering.

5.3.2 Wire Planning

A significant constraint for the floorplan and layout was the wire planning. Each channel required over two thousand wires going between the channel and the arbiters. As these wires are very long, care needs to be taken to avoid capacitive coupling. This was done in two ways: first, the wires were not driven with a very fast slew rate. As there is only one driver for each wire, and and the signals rise and fall monotonically, this will not affect the correctness. Secondly, the wires were spaced out. SPICE testing of extracted layout showed that this was superior to using power and ground to shield wires.

These connections needed to be arranged as shown in Figure 5.10 to implement the

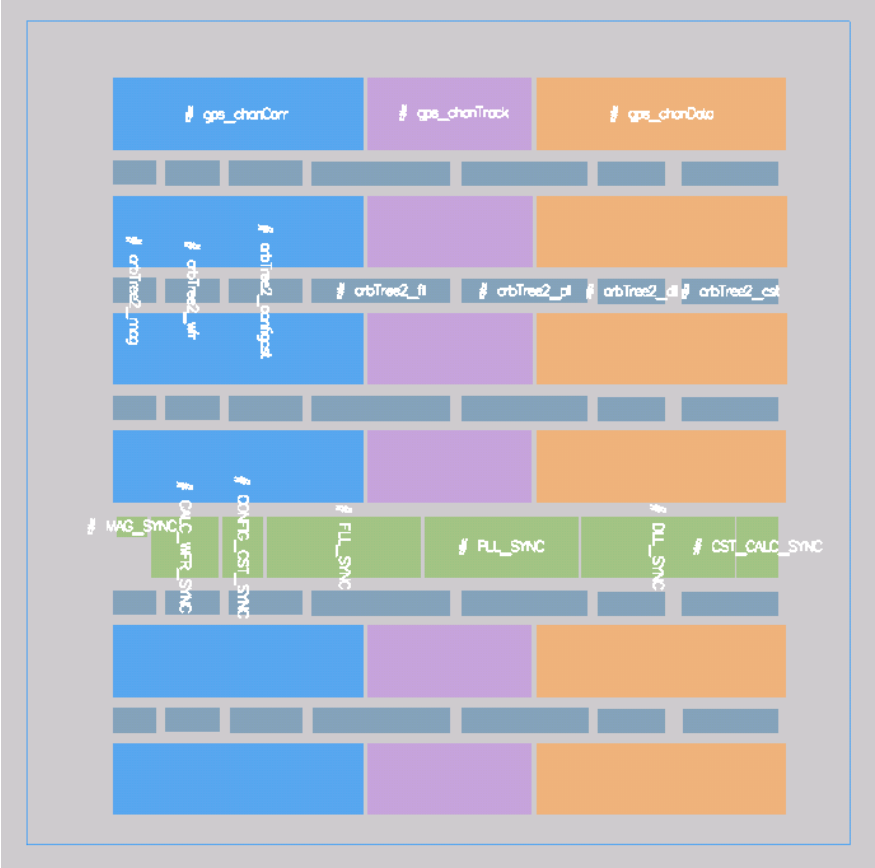

Figure 5.9: Image of the Approximate Chip Floorplan

arbitration tree. This was done using the organization shown in Figure 5.11. This plan ensures that the maximum load is never more than three times the minimum load for any path. This means that the approximately two thousand wires will need at least the space of six thousand wires. With a wire width of 0.18 microns, and a total width of 4 mm, this means that there will always be at least 0.487 microns of spacing available, which was found to be sufficient to avoid capacitive coupling.

The wiring load was most stressed at on the `chanTrack` block, where the number of transistors did not force the block into a width wide enough for all of the output wires. To get around that, some of the wires from the `chanData` module that were destined to the arbitration tree were moved to the left, and a bus turn from the left side of `chanTrack` was overlayed. This can be seen in Figure 5.8.

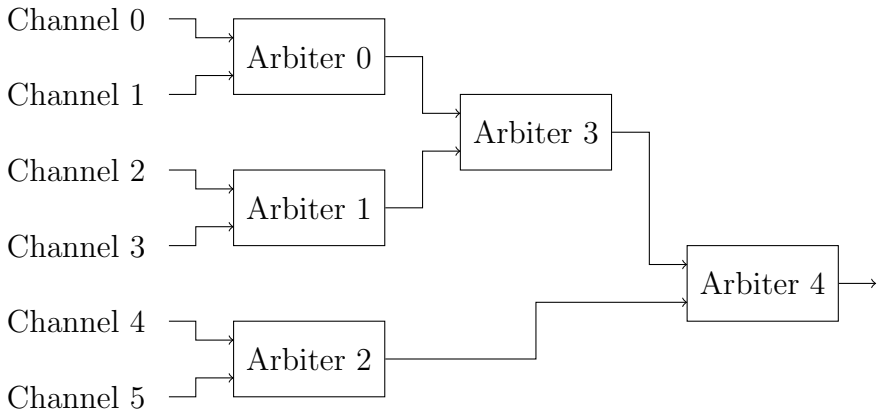

Figure 5.10: Arbitration Tree

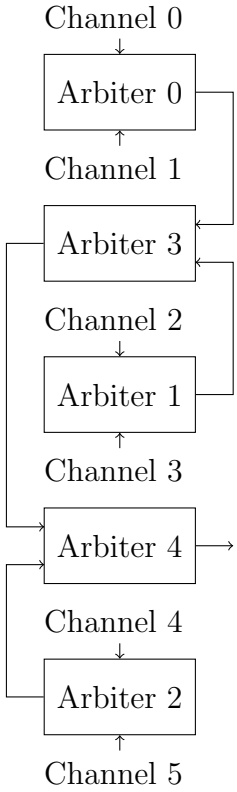

Figure 5.11: Arbitration Layout Plan

CHAPTER 6

PERFORMANCE

This section discusses how our system was simulated, and gives a comparison of our simulated results to those of other GPS receivers.

6.1 Receiver Performance Simulations

We first described the GPS system in the Communicating Hardware Processes (CHP) language, a variant of CSP that is widely used when describing QDI asynchronous circuits, discussed further in Appendix A. The CHP simulation was verified against a GPS software receiver written in MATLAB. Through the process of Martin synthesis [3], this high-level CHP description was translated into a gate-level description of our system. There are about 250K transistors per receiver channel. About 100K of these are in the fast-rate modules. The shared bundled-data math modules were synthesized from a Verilog description using commercial synthesis tools, resulting in approximately 570K transistors.

The correctness of the gate-level implementation is verified with a co-simulation involving Synopsys VCS and an in-house asynchronous circuit simulator known as PRSIM. The detailed simulation results—sequences of values transmitted on all communication channels in the system—from the gate-level co-simulation were found to match that of the CHP simulation.

To ensure sufficient testing coverage, we simulated the system with 60 seconds of satellite signals generated from the Spirent GPS simulator without atmospheric, ionospheric or multipath errors. The signal from the Spirent GPS simulator is fed into the Zarlink GP2015 front end providing digital samples with a sampling period of 175ns. These datapoints are recorded in a binary file that is subsequently fed into the CHP and gate-level simulations of our system. The receiver position is computed from 6 satellites and compared to the actual simulated position.

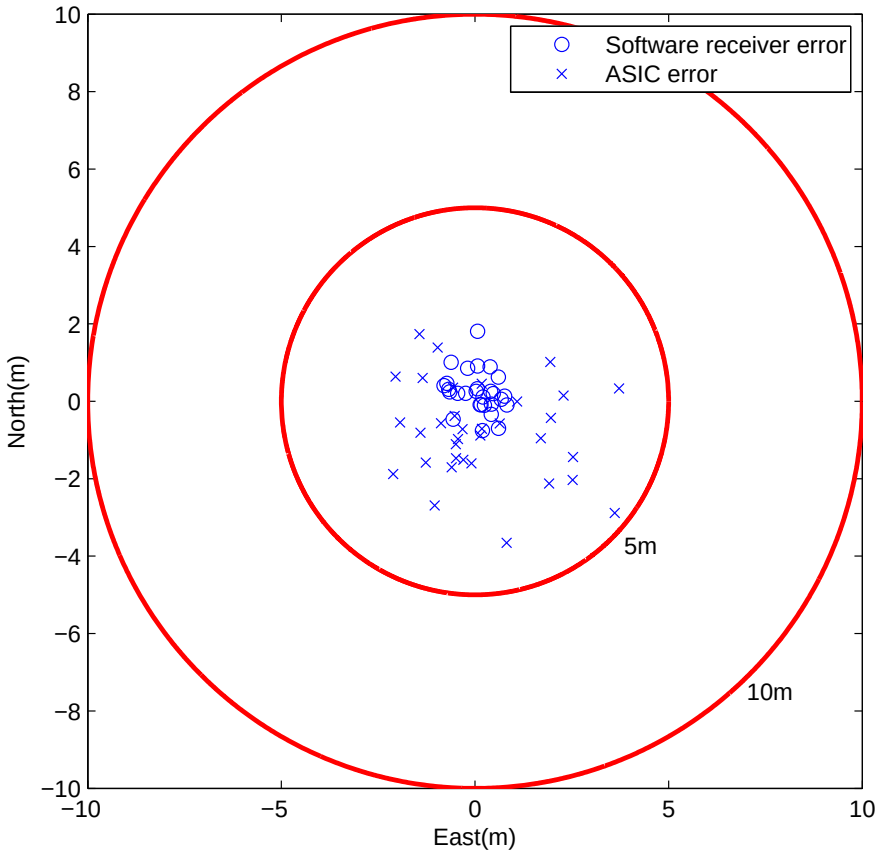

Figure 6.1: Position accuracy using 6 satellites

A comparison of position error between our system and a reference Matlab software receiver developed by the Cornell GPS Laboratory is shown in Figure 6.1. Our system has a larger error spread because of the use of a single-bit RF front end samples, longer acquisition, and from the delayed update, fixed point arithmetic, and arithmetic approximations used in the FLL, PLL and DLL tracking loops.

We tested tracking sensitivity with the Spirent GPS simulator using a graded reduction in carrier to noise ratio (C/No) every 10 seconds starting from the 30^{th} second. Figure 6.2 shows that for C/No above 35dB-Hz, the PLL experiences no loss of lock. Hence, the PLL performs well in normal conditions but in weak-signal environment, longer accumulation intervals would be needed to boost tracking sensitivity. Longer accumulations will translate

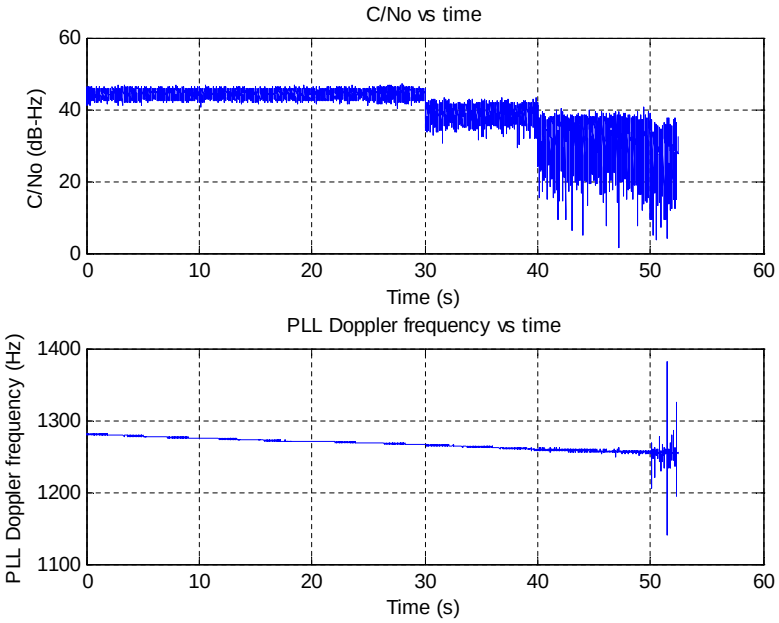

Figure 6.2: Tracking sensitivity test

to larger accumulation values. However, the advantage we have with our accumulator design is that we can increase the number of bits to represent the accumulator output without increasing much power consumption because the higher order bits rarely switch. If we were to increase our accumulation interval, we would need to change the width of the accumulators, and the design of the bundled-data math. As widening the accumulators corresponds to adding additional bits to the IDD unit, this will only marginally increase switching power. Additionally, as the frequency of the bundled-data arithmetic circuits is low, we can increase the width of this with only minimal cost to power. These increases would increase both the transistor usage and wire density, and may increase power loss through leakage if the system is manufactured in a different process technology.

The front end's ADC quantization noise also plays a part in tracking sensitivity. Figure 6.4 shows the in-phase and quadrature accumulations distribution when tracking with the PLL where the correlation power is concentrated in the in-phase portion of the signal and the data bit flips in the signal contribute to the 2 blobs on each side of the vertical axis.

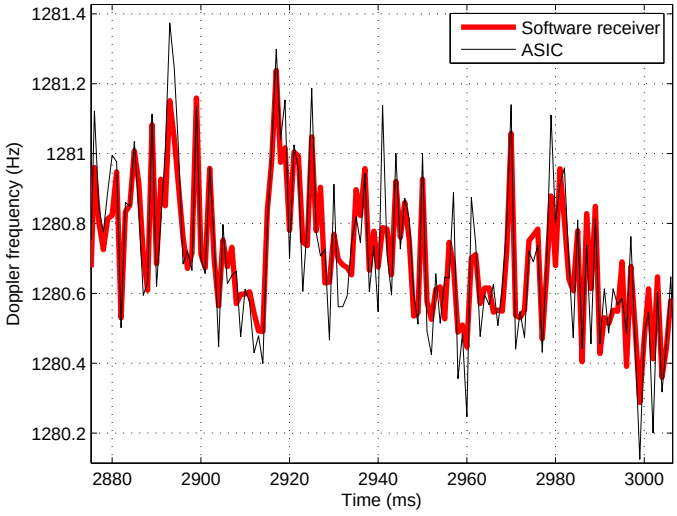

Figure 6.3: Tracking of Doppler frequency by the PLL compared to software receiver's

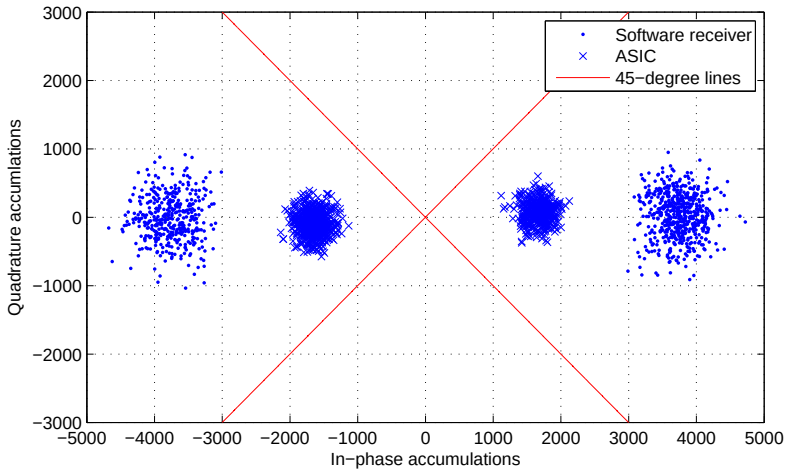

Figure 6.4: In-phase and quadrature accumulations phasor compared to software receiver's

The Matlab software receiver which uses 2-bit ADC samples has a higher correlation power than ours which uses only 1 bit. Despite using fixed-point arithmetic and approximations, our PLL tracks Doppler frequency relatively well, albeit with less precision than the Matlab software receiver as shown in Figure 6.3.

6.2 Power Simulations

We performed HSIM simulations on our system in a 90nm (Vdd=1V and T=25^{o}C) technology. To account for wire capacitance, we added wire load in the simulations to provide a more realistic account of our power numbers. The added wire capacitance is based on estimates from post-layout extractions of a portion of our system. Table 6.1 shows the power consumption of various modules in the system for 1 and all 6 receiver channels in acquisition and continuous tracking modes. These simulations were executed with a 5.714 MHz sampling frequency, and using the data from the Spirant GPS simulator.

Note that since the acquisition and tracking modes in our architecture share almost all of the fast-rate modules that make up the correlators, and they do not differ significantly in their behavior in real-world acquisition and tracking, the power consumption of the two modes is not significantly different. Since the implementation of a 6 receiver channel system involves duplicating the channel-dependent modules in a single channel six times, we can derive the power consumption of the full 6-channel system by adding the power consumption of the multiplexed shared math modules with 6 times the power consumption of the channel-dependent modules in a single channel. The total power consumed by our 6-channel system is approximately 1.5mW during acquisition and 1.4mW in continuous tracking mode.

Table 6.1: Power breakdown by module for a single channel and for six channels

	Acq (μW) (1 Chan)	Track (μW) (1 Chan)	Acq (μW) (6 Chan)	Track (μW) (6 Chan)
Acq control	12.8	10.1	77.0	61.0
Code Generator	6.96	6.66	41.8	39.9
Carrier NCO	79.6	73.8	477	443
Code NCO	73.2	66.7	439	400
6 Accumulators	61.2	59.9	367	360
Bundled-Data	3.95	4.09	5.90	6.38
Other	14.1	17.2	81.6	102.8
Total	252 μW	238 μW	1.49 mW	1.41 mW

Table 6.2: Power Comparison with State-of-the-Art

Name	This Work	[30]	[7]	[23]
Process (nm)	90	110	180	90
Voltage (V)	1.0	1.2	1.6	1.4
Number of Channels	6	22	12	-
System Power (mW)	1.4	34	56	84
RF Power (mW)	-	19.5	20	-
Baseband Power (mW)	1.4	14.5	36	-
Baseband Power/Channel (mW)	0.2	0.7	3	-
3-D rms Error (m)	3.9	-	3	-

The slowest shared bundled-data math module is the tracking loop math with a delay line of 250ns. Since each channel has an entire millisecond to obtain tracking loop updates, our architecture can technically be implemented for a system with up to 4000 channels sharing a single tracking loop math module, which is far greater than the minimum number of 4 channels and a typical number of 10 to 12. Simulations show that the power consumption of our system during static PLL tracking increases almost linearly with respect to the number of channels it supports because the accumulators and NCOs in each channel dominate the overall power consumption of the system.

Table 6.2 provides a comparison of our system with other contemporary GPS receivers. [30], [7] and [23] are system on chip (SoC) GPS receivers with integrated RF front end and digital baseband processing. To compare the per channel baseband processing power of our system with these SoCs, we subtract the power consumed by their RF front ends from their system power, the result of which is then divided by the number of channels tracked. Note that these SoCs have the ability to compute position estimates whereas our system stops short of doing that but provides measurements that can be used to derive position estimates in a host processor or base station. Nevertheless, the power consumed in position estimate computation is negligible for typical position update rates on the order of 1Hz. Taking all these into consideration, our per-channel baseband processing power is about 3 times lower than [30] and about 12 times lower than [7]. Our position estimates, computed offline, have

a 3D-RMS error below 4m which is comparable to [7] and is typical for receivers used in mobile devices. The higher number of satellites tracked by [7] however does improve its position accuracy due to the effects of Dilution of Precision (DOP) [18].

Table 6.3: Energy Comparison with State-of-the-Art

Name	This Work	[30]	[7]	[23]*	[19]	[12]*	BCM2075*
Process (nm)	90	110	180	90	500	-	-
Voltage (V)	1.0	1.2	1.6	1.4	1.5	-	1.2
Energy/Update	1.4 mJ	14.5 mJ	36 mJ	84 mJ	21.3 mJ	42 mJ	10 mJ
Scaled Energy	-	8.24 mJ	7.03 mJ	42.86 mJ	1.7 mJ	-	6.9 mJ

* Full system power used as breakdown not provided

As some of the current research into low-power GPS receivers has resulted in receivers with non-continuous output, it can be difficult to compare the power numbers of these to our design. Additionally, as commercial GPS systems do not provide much information about their implementation, only about their final results, it can be difficult to give a detailed breakdown. Table 6.3 instead compares the energy per baseband position generated between our design and others. This table also provides a scaled version of the energy, assuming that the power usage scales with the square of voltage, and linearly with process technology. This may overstate the improvements possible by scaling these systems.

In the cases where the breakdown between RF and Baseband power is not provided, we use the whole system power. For the non-continuously tracking systems, [19] [12], the energy is calculated assuming a 100% duty cycle for a single second, the typical interval between position updates. As these systems are less capable than the others presented, not collecting the received data, having lower potential accuracy and not being able to continuously track, these numbers may underestimate the effective energy used by a system involving them.

This table shows that our system uses less energy per position when compared to even the aggressively scaled version of the energy used by current state-of-the-art commercial and research devices.

CHAPTER 7

CONCLUSIONS

The design and implementation of an asynchronous low power GPS baseband processor has been presented. The system implements an acquisition scheme with sequential code phase search, a continuous tracking mode optimized for low power consumption and a data extraction scheme to provide synchronized Code Start Time (CST) measurements. The system achieves low power by allowing each subsystem to operate at its natural frequency, by reducing switching activity in the accumulators with an increment-decrement based design and by employing shared tracking loops with deferred updates, fixed-point arithmetic and mathematical approximations. Our system consumes 1.4mW during continuous tracking mode with position 3-D rms error below 4 meters.

Some of the techniques presented could benefit synchronous designers, by breaking their systems down into several clock domains. However, this transformation is more simply performed in asynchronous VLSI, particularly with the coprime 'clock' ratios used in GPS processors. They may be able to emulate some of the optimizations in the tracking loops, using shared resources and deferred updates, which may assist in decoupling performance. However, many of the low level optimizations were directly enabled by the asynchronous design methodology, and some of these (particularly the accumulator design) save significant amounts of energy.

The system presented here has been laid out for a 90 nm low power silicon transistor manufacturing process, and submitted to a production facility. It is expected to return in the near future, at which time the performance of the system will be tested. It is anticipated to closely match the simulated performance.

As future work, the system presented here can be extended to increase potential accuracy and ability in weak signal environments, or to make the acquisition procedure faster, possibly to enable reacquisition in the event of signal loss.

APPENDIX A

ASYNCHRONOUS HDL CONVENTIONS

Here we informally present the notation we use to describe QDI asynchronous circuits. A formal trace-tree based semantics can be found in [29].

A.1 CHP

Communicating Hardware Processes (CHP), is a variant of CSP [9], which has been found to be useful in representing high-level descriptions of asynchronous hardware.

As convention, variables are ranged over by lowercase letters, $a, b, c \ldots$, channels are ranged over by uppercase letters from the beginning of the alphabet, $A, B, C \ldots$, and programs or program segments are ranged over by uppercase letters from the second half of the alphabet, $P, Q, R \ldots$

- **Skip**: `skip`. This statement does nothing.
- **Assignment**: $x := E$. This statement means "assign the value of expression E to x." The statements $x\uparrow$ and $x\downarrow$ are shorthand for $x :=$ **true** and $x :=$ **false**, respectively.
- **Communication**: $A!e$ is a statement meaning "send the value of e over channel A," and $B?x$ means "receive a value over channel B and store it in variable x." Both sending and receiving are blocking.
- **Selection**: `[`$G_1 \rightarrow P_1$`[]`...`[]`$G_n \rightarrow P_n$`]`, where each G_i is a boolean expression, and each P_i is a statement. This statement is executed by waiting for one of the guards to be true, and then executing one statement with a true guard. If the guards are not mutually exclusive, we use the thin bar (`|`) instead of the thick bar (`[]`). `[`G`]` is used as a shorthand for `[`$G \rightarrow$ `skip]`.
- **Repetition**: `*[`$G_1 \rightarrow P_1$`[]`...`[]`$G_n \rightarrow P_n$`]`. This statement is executed by choosing one of the true guards and executing the corresponding statement, repeating until all guards

evaluate to false. If the guards are not mutually exclusive, we use the thin bar (|) instead of the thick bar ([]). *[P] is used as a shorthand for *[**true** $\rightarrow$ P].

- **Probe**: The boolean $\overline{A}$ is true if and only if a communication on channel A can complete without suspending. Probes are only allowed to occur in the guards of selection statements.
- **Sequential Composition**: $P; Q$. This statement means "execute statement P, and then execute statement Q."
- **Parallel Composition**:$P \parallel Q$. This statement means "execute statement P, while simultaneously executing statement Q."
- **Simultaneous Composition**: $P \star Q$. This statement means "execute channel actions P and Q such that they begin and complete simultaneously." This composition was introduced in [13] and replaces $\bullet$ composition of Martin [15]. Unlike the $\bullet$ composition, it is expressible in two-phase CHP, and will deadlock when composed in parallel with itself.

Weak Fairness Concurrent execution of CHP processes is assumed to be *weakly fair*, meaning that every continuously enabled action will eventually be given a chance to execute.

A.2 HSE

Handshaking Expansion (HSE) is a subset of CHP, where all actions on channels (communication, probes) are translated into Boolean variables, and all non-Boolean data representations are given Boolean representations.

A.3 PRS

A Production Rule Set (PRS) is a set of production rules, each of the form:

$$G \mapsto t$$

where t is a simple boolean assignment, and G is a boolean expression, known as the guard of the production rule. We require a few properties of a valid PRS:

- **Non-Interference** Production rules $G^+ \mapsto x\uparrow$ and $G^- \mapsto x\downarrow$ are said to be non-interfering in a computation if and only if $\neg G^+ \vee \neg G^-$ is an invariant of the computation. A valid PRS must only contain non-interfering rules, as interfering rules correspond to a short circuit.
- **Stability** A production rule $G \mapsto t$ is said to be stable in a computation if and only if G can change from true to false only after the assignment on t has completed. A valid QDI PRS will only contain stable production rules.

A PRS is executed as a state transformer, where any of the rules whose guards are satisfied by the current state may execute, modifying the state with their associated boolean assignment.

BIBLIOGRAPHY

[1] B. Alorda, G. Torrens, S. Bota, and J. Segura. Static-noise margin analysis during read operation of 6t sram cells. *DCIS Proceedings*, 2009.

[2] K. Borre, D. Akos, N. Bertelsen, P. Rinder, and S. Jensen. *A Software-Defined GPS and Galileo Receiver: A Single-Frequency Approach*. Birkhuser, Boston, MA, 2006.

[3] Janusz A. Brzozowski and Carl-Johan H. Seger. *Asynchronous circuits*. Monographs in computer science. Springer, 1995.

[4] Steven M Burns. *Performance Analysis and Optimization of Asynchronous Circuits*. PhD thesis, California Institute of Technology, December 1990.

[5] Virantha Ekanayake, Clinton Kelly, IV, and Rajit Manohar. An ultra low-power processor for sensor networks. *SIGOPS Oper. Syst. Rev.*, 38:27–36, October 2004.

[6] R. Gold. Optimal binary sequences for spread spectrum multiplexing (corresp.). *Information Theory, IEEE Transactions on*, 13(4):619 –621, october 1967.

[7] G. Gramegna, P.G. Mattos, M. Losi, S. Das, M. Franciotta, N.G. Bellantone, M. Vaiana, V. Mandara, and M. Paparo. A 56-mw 23-mm2 single-chip 180-nm cmos gps receiver with 27.2-mw 4.1-mm2 radio. *Solid-State Circuits, IEEE Journal of*, 41(3):540 – 551, march 2006.

[8] A. Heiberg, T. Brown, K. Mayaram, and T.S. Fiez. A 250 mv, 352 μ;w low-if quadrature gps receiver in 130 nm cmos. In *VLSI Circuits (VLSIC), 2010 IEEE Symposium on*, pages 135 –136, june 2010.

[9] C.A.R. Hoare. Communicating sequential processes. In *Communications of the ACM*, volume 21, pages 666–677, August 1978.

[10] E. Kaplan and C. Hegarty. *Understanding GPS: Principles and Applications, Second Edition*. Artech House, Norwood, MA, 2005.

[11] Tony Lee. *ergen man page*, 1.0 edition, May 1994.

[12] Jie Liu, Bodhi Priyantha, Ted Hart, Heitor S. Ramos, Antonio A.F. Loureiro, and Qiang Wang. Energy efficient gps sensing with cloud offloading. In *10th ACM Conference on Embedded Networked Sensor Systems*, November 2012.

[13] R. Manohar. An analysis of reshuffled handshaking expansions. In *Asynchronous Circuits and Systems, 2001. ASYNC 2001. Seventh International Symposium on*, pages 96 –105, 2001.

[14] A.J. Martin. Remarks on low-power advantages of asynchronous circuits. In *Europ. Solid-State Circuits Conf.(ESSCIRC)*, 1998.

[15] Alain J. Martin. Compiling communicating processes into delay-insensitive VLSI circuits. *Distributed Computing*, 1(4):226–234, 1986.

[16] Alain J. Martin. A synthesis method for self-timed VLSI circuits. In *Proc. International Conf. Computer Design (ICCD)*, pages 224–229, Rye Brook, NY, 1987. IEEE Computer Society Press.

[17] T.H. Meng. Low-power gps receiver design. In *Signal Processing Systems, 1998. SIPS 98. 1998 IEEE Workshop on*, pages 1 –10, oct 1998.

[18] P. Misra and P. Enge. *Global Positioning System: Signals, Measurements, and Performance*. Ganga-Jamuna Press, Lincoln, MA, 2006.

[19] Won Namgoong, S. Reader, and T.H. Meng. An all-digital low-power if gps synchronizer. *Solid-State Circuits, IEEE Journal of*, 35(6):856 –864, jun 2000.

[20] NOAA. Gps.gov: Selective availability. http://www.gps.gov/systems/gps/modernization/sa/. Online.

[21] B.W. Parkinson and J.J. Spilker. *The global positioning system: theory and applications*. Number v. 1; v. 163 in Progress in astronautics and aeronautics. American Institute of Aeronautics and Astronautics, 1996.

[22] Cheryl Pellerin. United states updates global positioning system technology. http://www.america.gov/st/washfile-english/2006/February/20060203125928lcnirellep0.5061609.html, February 2006. Online.

[23] D. Sahu, A. Das, Y. Darwhekar, S. Ganesan, G. Rajendran, R. Kumar, B.G. Chandrashekar, A. Ghosh, A. Gaurav, T. Krishnaswamy, A. Goyal, S. Bhagavatheeswaran, Kah Mun Low, N. Yanduru, S. Dhamankar, and S. Venkatraman. A 90nm cmos single-chip gps receiver with 5dbm out-of-band iip3 2.0db nf. In *Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International*, pages 308 –600 Vol. 1, feb. 2005.

[24] E. Seevinck, F.J. List, and J. Lohstroh. Static-noise margin analysis of mos sram cells. *Solid-State Circuits, IEEE Journal of*, 22(5):748–754, 1987.

[25] B.R. Sheikh and R. Manohar. An operand-optimized asynchronous ieee 754 double-precision floating-point adder. In *Asynchronous Circuits and Systems (ASYNC), 2010 IEEE Symposium on*, pages 151 –162, may 2010.

[26] B.Z. Tang, S. Longfield, S.A. Bhave, and R. Manohar. A low power asynchronous gps baseband processor. In *Asynchronous Circuits and Systems (ASYNC), 2012 18th IEEE International Symposium on*, pages 33 –40, may 2012.

[27] John Teifel and Rajit Manohar. Static tokens: Using dataflow to automate concurrent pipeline synthesis. In *In Proceedings of International Symposium on Asynchronous Circuits and Systems*, pages 17–27, 2004.

[28] K. Van Berkel. *Handshake Circuits: An Asynchronous Architecture for Vlsi Programming*. Cambridge International Series on Parallel Computation : 5. Cambridge University Press, 1993.

[29] M van der Goot. *Semantics of VLSI Synthesis*. PhD thesis, California Institute of Technology, May 1995.

[30] J.-M. Wei, C.-N. Chen, K.-T. Chen, C.-F. Kuo, B.-H. Ong, C.-H. Lu, C.-C. Liu, H.-C. Chiou, H.-C. Yeh, J.-H. Shieh, K.-S. Huang, K.-I. Li, M.-J. Wu, M.-H. Li, S.-H. Chou, S.-L. Chew, W.-L. Lien, W.-G. Yau, W.-Z. Ge, W.-C. Lai, W.-H. Ting, Y.-J. Tsai, Y.-C. Yen, and Y.-C. Yeh. A 110nm rfcmos gps soc with 34mw -165dbm tracking sensitivity. In *Solid-State Circuits Conference - Digest of Technical Papers, 2009. ISSCC 2009. IEEE International*, pages 254 –255,255a, feb. 2009.